



# VSource interface C++ library

---

v1.7.1

## Table of contents

---

- [Overview](#)
- [Versions](#)
- [Library files](#)
- [Video source interface class description](#)
  - [VSource class declaration](#)
  - [getVersion method](#)
  - [openVSource method](#)
  - [initVSource method](#)
  - [isVSourceOpen method](#)
  - [closeVSource method](#)
  - [getFrame method](#)
  - [setParam method](#)
  - [getParam method](#)
  - [getParams method](#)
  - [executeCommand method](#)
  - [encodeSetParamCommand method](#)
  - [encodeCommand method](#)
  - [decodeCommand method](#)
  - [decodeAndExecuteCommand method](#)
- [Data structures](#)
  - [VSourceCommand enum](#)
  - [VSourceParam enum](#)
- [VSourceParams class description](#)
  - [VSourceParams class declaration](#)
  - [Serialize video source params](#)
  - [Deserialize video source params](#)
  - [Read params from JSON file and write to JSON file](#)
- [Build and connect to your project](#)

- [How to make custom implementation](#)

## Overview

**VSource** C++ library provides standard interface as well defines data structures and rules for different video source classes (video capture classes). **VSource** interface class doesn't do anything, just provides interface and provides methods to encode/decode commands and encode/decode params. Also **VSource** class provides data structures for video source parameters. Different video source classes inherit interface from **VSource** C++ class. **VSource.h** file contains contains list of data structures ([VSourceCommand enum](#), [VSourceParam enum](#) and [VSourceParams class](#) class). [VSourceParams class](#) contains video source params and includes methods to encode and decode video source params. [VSourceCommand enum](#) contains IDs of commands supported by **VSource** class. [VSourceParam enum](#) contains IDs of params supported by **VSource** class. All video sources should include params and commands listed in **VSource.h** file. **VSource** class interface class depends on [Frame](#) class (describes video frame and video frame data structures, necessary for autofocus functions) and [ConfigReader](#) library (provides methods to read/write JSON config files).

## Versions

**Table 1** - Library versions.

| Version | Release date | What's new   |
|---------|--------------|--|
| 1.0.0   | 13.06.2023   | First version  |
| 1.0.2   | 20.06.2023   | - Fixed bugs.<br>- Documentation updated.  |
| 1.1.0   | 29.06.2023   | - Added new parameters.<br>- Documentation updated.  |
| 1.1.1   | 29.06.2023   | - Added license.<br>- Repository made public.  |
| 1.2.0   | 01.07.2023   | - Added new methods to encode/decode commands.<br>- Tests updated.<br>- Documentation updated.                         |
| 1.2.1   | 01.07.2023   | - Params description updated in source code.   |
| 1.3.0   | 05.07.2023   | - VSourceParams class updated (initString replaced by source).<br>- decode(...) method in VSourceParams class updated. |
| 1.3.1   | 06.07.2023   | - gain, exposure and focusPos fields of VSourceParams excluded from JSOM reading/writing.                              |
| 1.3.2   | 06.07.2023   | - Frame library version updated.   |

| Version | Release date | What's new   |
|---------|--------------|--|
| 1.4.0   | 11.07.2023   | - Added VSourceParamsMask structure.<br>- Added params mask in encode(...) method of VSourceParams class.  |
| 1.5.0   | 22.09.2023   | - Updated encode(...) and decode(...) methods of VSourceParams.<br>- Added decodeAndExecuteCommand(...) method.<br>- Added example of video source implementation. |
| 1.6.0   | 26.09.2023   | - Signature of getParams(...) method changed.  |
| 1.6.1   | 13.11.2023   | - Frame class updated.   |
| 1.7.1   | 13.12.2023   | - Virtual destructor added.  |

## Library files

The **VSource** library is a CMake project. Library files:

```

CMakeLists.txt ----- Main CMake file of the library.
3rdparty ----- Folder with third-party libraries.
    CMakeLists.txt ----- CMake file which includes third-party libraries.
    ConfigReader ----- Source code of the ConfigReader library.
    Frame ----- Source code of the Frame library.
example ----- Folder with simple example of VCodecImSdk usage.
    CMakeLists.txt ----- CMake file for example custom video source class.
    CustomVSource.cpp ----- Source code file of the CustomVSource class.
    CustomVSource.h ----- Header with CustomVSource class declaration.
    CustomVSourceVersion.h ----- Header file which includes CustomVSource version.
    CustomVSourceVersion.h.in ---- CMake service file to generate version file.
test ----- Folder with codec test application.
    CMakeLists.txt ----- CMake file for codec test application.
    main.cpp ----- Source code file of VSource class test application.
src ----- Folder with source code of the library.
    CMakeLists.txt ----- CMake file of the library.
    VSource.cpp ----- Source code file of the library.
    VSource.h ----- Header file which includes VSource class declaration.
    VSourceVersion.h ----- Header file which includes version of the library.
    VSourceVersion.h.in ----- CMake service file to generate version file.

```

## Video source interface class description

### VSource class declaration

**VSource** interface class declared in **VSource.h** file. Class declaration:

```

class VSource
{
public:

    /// Class destructor.
    virtual ~VSource();

    /// Get string of current library version.
    static std::string getVersion();

    /// Open video source.
    virtual bool openVSource(std::string& initString) = 0;

    /// Init video source.
    virtual bool initVSource(VSourceParams& params) = 0;

    /// Get open status.
    virtual bool isVSourceOpen() = 0;

    /// Close video source.
    virtual void closeVSource() = 0;

    /// Get new video frame.
    virtual bool getFrame(Frame& frame, int32_t timeoutMsec = 0) = 0;

    /// Set video source param.
    virtual bool setParam(VSourceParam id, float value) = 0;

    /// Get video source param value.
    virtual float getParam(VSourceParam id) = 0;

    /// Get video source params structure.
    virtual void getParams(VSourceParams& params) = 0;

    /// Execute command.
    virtual bool executeCommand(VSourceCommand id) = 0;

    /// Encode set param command.
    static void encodeSetParamCommand(
        uint8_t* data, int& size, VSourceParam id, float value);

    /// Encode command.
    static void encodeCommand(
        uint8_t* data, int& size, VSourceCommand id);

    /// Decode command.
    static int decodeCommand(uint8_t* data,
        int size,
        VSourceParam& paramId,
        VSourceCommand& commandId,
        float& value);

    /// Decode and execute command.
    virtual bool decodeAndExecuteCommand(uint8_t* data, int size) = 0;
};

```

## getVersion method

**getVersion()** method returns string of current version of **VSource** class. Particular video source class can have it's own **getVersion()** method. Method declaration:

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

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

```
std::cout << "VSource class version: " << VSource::getVersion() << std::endl;
```

Console output:

```
VSource class version: 1.7.1
```

## openVSource method

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

```
virtual bool openVSource(std::string& initString) = 0;
```

| Parameter  | Value  |
|------------|--|
| initString | Initialization string. Format depends on implementation but it is recommended to keep default format: [video device or ID or file];[width];[height];[fourcc]. Example: "/dev/video0;1920;1080;YUYV". |

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

## initVSource method

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

```
virtual bool initVSource(VSourceParams& params) = 0;
```

| Parameter | Value   |
|-----------|---|
| params    | VSourceParams structure (see <a href="#">VSourceParams class</a> description). The video source should set parameters according to params structure. Particular video source can support not all parameters listed in <a href="#">VSourceParams class</a> . |

**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:

```
virtual bool isVSourceOpen() = 0;
```

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

## closeVSource method

**closeVSource()** method closes video source. Method declaration:

```
virtual void closeVSource() = 0;
```

## getFrame method

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

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

| Parameter   | Value  |
|-------------|--|
| frame       | Output video frame (see <a href="#">Frame</a> class description). Video source class determines output pixel format. Pixel format can be set in <b>initVSource(...)</b> or <b>openVSource(...)</b> methods if particular video source supports it.   |
| timeoutMsec | Timeout to wait new frame data: <ul style="list-style-type: none"><li>- timeoutMs == -1 - Method will wait endlessly until new data arrive.</li><li>- timeoutMs == 0 - Method will only check if new data exist.</li><li>- timeoutMs &gt; 0 - Method will wait new data specified time.</li></ul> <b>Each video source implementation must provide described behavior.</b> |

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

## setParam method

**setParam(...)** method sets new video source parameters value. The particular implementation of the video source must provide thread-safe **setParam(...)** method call. This means that the **setParam(...)** method can be safely called from any thread. Method declaration:

```
virtual bool setParam(VSourceParam id, float value) = 0;
```

| Parameter | Description  |
|-----------|--|
| id        | Video source parameter ID according to <a href="#">VSourceParam enum</a> . |
| 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. The particular implementation of the video source must provide thread-safe **getParam(...)** method call. This means that the **getParam(...)** method can be safely called from any thread. Method declaration:

```
virtual float getParam(VSourceParam id) = 0;
```

| Parameter | Description  |
|-----------|--|
| id        | Video source parameter ID according to <a href="#">VSourceParam enum</a> . |

**Returns:** parameter value or -1 if the parameter doesn't exist in particular video source class.

## getParams method

**getParams(...)** method designed to obtain video source params structure. The particular implementation of the video source must provide thread-safe **getParams(...)** method call. This means that the **getParams(...)** method can be safely called from any thread. Method declaration:

```
virtual void getParams(VSourceParams& params) = 0;
```

| Parameter | Description                                       |
|-----------|---|
| params    | Video source params class object (VSourceParams). |

## executeCommand method

**executeCommand(...)** method designed to execute video source command. The particular implementation of the video source must provide thread-safe **executeCommand(...)** method call. This means that the **executeCommand(...)** method can be safely called from any thread. Method declaration:

```
virtual bool executeCommand(VSourceCommand id) = 0;
```

| Parameter | Description  |
|-----------|--|
| id        | Video source command ID according to <a href="#">VSourceCommand enum</a> . |

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

## encodeSetParamCommand method

**encodeSetParamCommand(...)** static method 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 <a href="#">VSourceParam enum</a> .     |
| 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    | 0x06  | Minor version of VSource class.                       |
| 3    | id    | Parameter ID <b>int32_t</b> in Little-endian format.  |
| 4    | id    | Parameter ID <b>int32_t</b> in Little-endian format.  |
| 5    | id    | Parameter ID <b>int32_t</b> in Little-endian format.  |
| 6    | id    | Parameter ID <b>int32_t</b> in Little-endian format.  |
| 7    | value | Parameter value <b>float</b> in Little-endian format. |
| 8    | value | Parameter value <b>float</b> in Little-endian format. |
| 9    | value | Parameter value <b>float</b> in Little-endian format. |
| 10   | value | Parameter value <b>float</b> 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.
VSource::encodeSetParamCommand(data, size, VSourceParam::EXPOSURE, outValue);

```

## encodeCommand method

**encodeCommand(...)** static method 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:

```

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

```

| Parameter | Description  |
|-----------|--|
| data      | Pointer to data buffer for encoded command. Must have size >= 7 bytes. |
| size      | Size of encoded data. Will be 7 bytes.                                 |
| id        | Command ID according to <a href="#">VSourceCommand enum</a> .          |

**COMMAND** format:

| Byte | Value | Description  |
|------|-------|--|
| 0    | 0x00  | COMMAND header value.                              |
| 1    | 0x01  | Major version of VSource class.                    |
| 2    | 0x06  | Minor version of VSource class.                    |
| 3    | id    | Command ID <b>int32_t</b> in Little-endian format. |
| 4    | id    | Command ID <b>int32_t</b> in Little-endian format. |
| 5    | id    | Command ID <b>int32_t</b> in Little-endian format. |
| 6    | id    | Command ID <b>int32_t</b> in Little-endian format. |

**encodeCommand(...)** 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;
// Encode command.
VSource::encodeCommand(data, size, VSourceCommand::RESTART);
```

## decodeCommand method

**decodeCommand(...)** static method 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 for SET_PARAM and 7 bytes for COMMAND.  |
| paramId   | Parameter ID according to <a href="#">VSourceParam enum</a> . After decoding SET_PARAM command the method will return parameter ID. |
| commandId | Command ID according to <a href="#">VSourceCommand enum</a> . 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.

## decodeAndExecuteCommand method

**decodeAndExecuteCommand(...)** method decodes and executes command on video source side. The particular implementation of the video source must provide thread-safe **decodeAndExecuteCommand(...)** method call. This means that the **decodeAndExecuteCommand(...)** method can be safely called from any thread. Method declaration:

```
virtual bool decodeAndExecuteCommand(uint8_t* data, int size) = 0;
```

| Parameter | Description  |
|-----------|--|
| data      | Pointer to input command.  |
| size      | Size of command. Must be 11 bytes for SET_PARAM and 7 bytes for COMMAND. |

**Returns:** TRUE if command decoded (SET\_PARAM or COMMAND) and executed (action command or set param command).

# Data structures

**VSource.h** file 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 | Restart video source (close and open again). |

## VSourceParam enum

Enum declaration:

```
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 | Frame width. User can set frame width before initialization or after. Some video source classes may set width automatically.    |
| HEIGHT        | read / write | Frame height. User can set frame height before initialization or after. Some video source classes may set height automatically. |
| GAIN_MODE     | read / write | Gain mode. Value depends on implementation but it is recommended to keep default values: 0 - Manual control, 1 - Auto.          |
| GAIN          | read / write | Gain value. Value: 0(min for particular video source class) - 65535(max for particular video source class).                     |
| EXPOSURE_MODE | read / write | Exposure mode. Value depends on implementation but it is recommended to keep default values: 0 - Manual control, 1 - Auto.      |
| EXPOSURE      | read / write | Exposure value. Value: 0(min for particular video source class) - 65535(max for particular video source class).                 |
| FOCUS_MODE    | read / write | Focus mode. Value depends on implementation but it is recommended to keep default values: 0 - Manual control, 1 - Auto.         |
| FOCUS_POS     | read / write | Focus position. Value: 0(full near) - 65535(full far).  |

| Parameter      | Access       | Description  |
|----------------|--------------|--|
| CYCLE_TIME_MKS | read only    | Video capture cycle time. <b>VSource</b> class sets this value automatically. This parameter means time interval between two captured video frame. |
| FPS            | read / write | FPS. User can set frame FPS before initialization or after. Some video source classes may set FPS automatically.                                   |
| IS_OPEN        | read only    | Open flag. 0 - not open, 1 - open.   |
| CUSTOM_1       | read / write | Custom parameter. Depends on implementation.   |
| CUSTOM_2       | read / write | Custom parameter. Depends on implementation.   |
| CUSTOM_3       | read / write | Custom parameter. Depends on implementation.   |

## 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:

```
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};
    /// 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, exposureMode,
                  focusMode, fps, custom1, custom2, custom3);

    /// operator =
    VSourceParams& operator= (const VSourceParams& src);

    /// Encode params.
    bool encode(uint8_t* data, int bufferSize, int& size,
                VSourceParamsMask* mask = nullptr);

    /// Decode params.
    bool decode(uint8_t* data, int dataSize);
};

```

**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. |

| Field        | type   | Description   |
|--------------|--------|---|
| source       | string | Video source: file, video stream, video device, camera num, etc.  |
| fourcc       | string | 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. |
| width        | int    | Frame width. User can set frame width before initialization or after. Some video source classes may set width automatically.  |
| height       | int    | Frame height. User can set frame height before initialization or after. Some video source classes may set height automatically.   |
| gainMode     | int    | Gain mode. Value depends on implementation but it is recommended to keep default values: 0 - Manual control, 1 - Auto.  |
| gain         | int    | Gain value. Value: 0(min for particular video source class) - 65535(max for particular video source class).   |
| exposureMode | int    | Exposure mode. Value depends on implementation but it is recommended to keep default values: 0 - Manual control, 1 - Auto.  |
| exposure     | int    | Exposure value. Value: 0(min for particular video source class) - 65535(max for particular video source class).   |
| focusMode    | int    | Focus mode. Value depends on implementation but it is recommended to keep default values: 0 - Manual control, 1 - Auto.   |
| focusPos     | int    | Focus position. Value: 0(full near) - 65535(full far).  |
| cycleTimeMks | int    | Video capture cycle time. <b>VSource</b> class sets this value automatically. This parameter means time interval between two captured video frame.  |
| fps          | float  | FPS. User can set frame FPS before initialization or after. Some video source classes may set FPS automatically.  |
| isOpen       | bool   | Open flag. false - not open, true - open.   |
| custom1      | float  | Custom parameter. Depends on implementation.  |
| custom2      | float  | Custom parameter. Depends on implementation.  |
| custom3      | float  | Custom parameter. Depends on implementation.  |

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

## Serialize 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 provides options to exclude particular parameters from serialization. To do this method inserts binary mask (2 bytes) where each bit represents particular parameter and **decode(...)** method recognizes it. Method declaration:

```
bool encode(uint8_t* data, int bufferSize, int& size, VSourceParamsMask* mask = nullptr);
```

| Parameter  | Value   |
|------------|---|
| data       | Pointer to data buffer. Buffer size should be at least <b>62</b> bytes.   |
| size       | Size of encoded data. 62 bytes by default.  |
| bufferSize | Data buffer size. Buffer size must be >= 62 bytes.  |
| mask       | Parameters mask - pointer to <b>VSourceParamsMask</b> structure. <b>VSourceParamsMask</b> (declared in VSource.h file) determines flags for each field (parameter) declared in <b>VSourceParams</b> class. If the user wants to exclude any parameters from serialization, he can put a pointer to the mask. If the user wants to exclude a particular parameter from serialization, he should set the corresponding flag in the VSourceParamsMask structure. |

**VSourceParamsMask** structure declaration:

```
typedef struct VSourceParamsMask
{
    bool logLevel{true};
    bool width{true};
    bool height{true};
    bool gainMode{true};
    bool gain{true};
    bool exposureMode{true};
    bool exposure{true};
    bool focusMode{true};
    bool focusPos{true};
    bool cycleTimeMks{true};
    bool fps{true};
    bool isOpen{true};
    bool custom1{true};
    bool custom2{true};
    bool custom3{true};
} VSourceParamsMask;
```

Example without parameters mask:



```
// Prepare random params.
VSourceParams in;
in.initString = "alsfghljb";
in.logLevel = 0;

// Encode data.
uint8_t data[1024];
int size = 0;
in.encode(data, 1024, size);
cout << "Encoded data size: " << size << " bytes" << endl;
```

Example without parameters mask:

```
// Prepare random params.
VSourceParams in;
in.initString = "alsfghljb";
in.logLevel = 0;

// Prepare params mask.
VSourceParamsMask mask;
mask.logLevel = false; // Exclude logLevel. Others by default.

// Encode data.
uint8_t data[1024];
int size = 0;
in.encode(data, 1024, size, &mask);
cout << "Encoded data size: " << size << " bytes" << endl;
```

## Deserialize 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 automatically recognizes which parameters were serialized by **encode(...)** method. Method declaration:

```
bool decode(uint8_t* data, int dataSize);
```

| Parameter | Value  |
|-----------|--|
| data      | Pointer to encode data buffer. Data size should be at least <b>62</b> bytes. |
| dataSize  | Size of data.  |

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

Example:

```

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

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

// Decode data.
VSourceParams out;
if (!out.decode(data, size))
    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:

```

// Write params to file.
VSourceParams 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:

```

{
    "vSourceParams": {
        "custom1": 96.0,
        "custom2": 212.0,
        "custom3": 243.0,
        "exposureMode": 63,
        "focusMode": 167,
        "fourcc": "skdfjhvk",
        "fps": 205.0,
        "gainMode": 84,
        "height": 143,
        "logLevel": 92,
        "source": "alsfghljb",
        "width": 204
    }
}

```

# Build and connect to your project

Typical commands to build **VSource** library:

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

If you want connect **VSource** 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 **VSource** as submodule by commands:

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

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

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

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(${PARENT}_SUBMODULE_VSOURCE ON CACHE BOOL "" FORCE)
if (${PARENT}_SUBMODULE_VSOURCE)
    SET(${PARENT}_VSOURCE ON CACHE BOOL "" FORCE)
    SET(${PARENT}_VSOURCE_TEST OFF CACHE BOOL "" FORCE)
    SET(${PARENT}_VSOURCE_EXAMPLE OFF CACHE BOOL "" FORCE)
endif()

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

```

File **3rdparty/CMakeLists.txt** adds folder **VSource** to your project and excludes test application and example (VSource class test applications and example of custom video source class implementation) from compiling. Your repository new structure will be:

```

CMakeLists.txt
src
    CMakeList.txt
    yourLib.h
    yourLib.cpp
3rdparty
    CMakeLists.txt
    VSource

```

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 VSource library in your **src/CMakeLists.txt** file:

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

Done!

# How to make custom implementation

The **VSource** class provides only an interface, data structures, and methods for encoding and decoding commands and params. To create your own implementation of the video source, you must include the VSource repository in your project (see [Build and connect to your project](#) section). The catalogue **example** (see [Library files](#) section) includes an example of the design of the custom video source. You must implement all the methods of the VSource interface class. Custom video source class declaration:

```
class CustomVSource: public VSource
{
public:

    /// Class constructor.
    CustomVSource();

    /// Class destructor.
    ~CustomVSource();

    /// Get string of current library version.
    static std::string getVersion();

    /// Open video source.
    bool openVSource(std::string& initString);

    /// Init video source.
    bool initVSource(VSourceParams& params);

    /// Get open status.
    bool isVSourceOpen();

    /// Close video source.
    void closeVSource();

    /// Get new video frame.
    bool getFrame(Frame& frame, int32_t timeoutMsec = 0);

    /// Set video source param.
    bool setParam(VSourceParam id, float value);

    /// Get video source param value.
    float getParam(VSourceParam id);

    /// Get video source params structure.
    void getParams(VSourceParams& params);

    /// Execute command.
    bool executeCommand(VSourceCommand id);

    /// Decode and execute command.
    bool decodeAndExecuteCommand(uint8_t* data, int size);

private:

    /// video source params.
```

```
VSourceParams m_params;  
/// Output frame.  
Frame m_outputFrame;  
};
```