VSource interface C++ library

v1.4.0

Table of contents

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. Also **VSource** class provides data structures for video source parameters. Different video source classes inherit interface form **VSource** C++ 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 includes 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. All video sources should include params and commands listed in **VSource.h** file. VSource class dependency:
- **Frame** class which describes video frame structure and pixel formats.
- **ConfigReader** class which provides methods to work with JSON structures (read/write).

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. - Documentation updated.
1.1.0	29.06.2023	- Added new parameters. - Documentation updated.
1.1.1	29.06.2023	- Added license. - Repository made public.
1.2.0	01.07.2023	- Added new methods to encode/decode commands. - Tests updated. - 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). - 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.
1.4.0	11.07.2023	- Added VSourceParamsMask structure. - Added params mask in encode(...) method of VSourceParams class.

Video source interface class description

VSource class declaration

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

```
class VSource
{
public:
    /**
     * @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 depends on implementation.
     * Default format: <video device or ID or file>;<width>;<height>;<fourcc>
     * @return TRUE if the video source open or FALSE if not.
     */
    virtual bool openVSource(std::string& initString) = 0;
    /**
     * @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.
     */
    virtual bool initVSource(VSourceParams& params) = 0;
    /**
     * @brief Get open status.
     * @return TRUE if video source open or FALSE if not.
     */
    virtual bool isVSourceOpen() = 0;
    /**
     * @brief Close video source.
     */
    virtual void closeVSource() = 0;
    /**
     * @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.
     */
    virtual bool getFrame(Frame& frame, int32_t timeoutMsec = 0) = 0;
    /**
     * @brief Set video source param.
     * @param id Parameter ID.
     * @param value Parameter value to set.
     * @return TRUE if property was set of FALSE.
     */
```

```
    virtual bool setParam(VSourceParam id, float value) = 0;
    /**
     * @brief Get video source param value.
     * @param id Parameter ID.
     * @return Parameter value or -1.
     */
    virtual float getParam(VSourceParam id) = 0;
    /**
     * @brief Get video source params structure.
     * @return Video source parameters structure.
     */
    virtual VSourceParams getParams() = 0;
    /**
     * @brief Execute command.
     * @param id Command ID.
     * @return TRUE if the command accepted or FALSE if not.
     */
    virtual bool executeCommand(VSourceCommand id) = 0;
    /**
     * @brief Encode set param command.
     * @param data Pointer to data buffer. Must have size >= 11.
     * @param size Size of encoded data.
     * @param id Parameter id.
     * @param value Parameter value.
     */
    static void encodeSetParamCommand(
            uint8_t* data, int& size, VSourceParam id, float value);
    /**
     * @brief Encode command.
     * @param data Pointer to data buffer. Must have size >= 11.
     * @param size Size of encoded data.
     * @param id Command ID.
     */
    static void encodeCommand(
            uint8_t* data, int& size, VSourceCommand id);
    /**
     * @brief Decode command.
     * @param data Pointer to command data.
     * @param size Size of data.
     * @param paramId Output command ID.
     * @param commandId Output command ID.
     * @param value Param or command value.
     * @return 0 - command decoded, 1 - set param command decoded, -1 - error.
     */
    static int decodeCommand(uint8_t* data,
                             int size,
                             VSourceParam& paramId,
                             VSourceCommand& commandId,
                             float& value);
};
```

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: 1.4.0
```

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 designed to initialize 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 **VSourceParams** class description). The video source should set parameters according to params structure. Particular video source can support not all parameters listed in VSourceParams class.

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 intended to close 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 **Frame** class description). Video source class determines output pixel format. Pixel format can be set in **initVSource(...)** or **openVSource(...)** methods if particular video source supports it.
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. **Each video source implementation must provide described behavior.**

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 declaration:

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

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

Parameter	Description
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 declaration:

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

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

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

getParams method

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

```
virtual VSourceParams getParams() = 0;
```

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

executeCommand method

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

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

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

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 **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

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 >= 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:

```
// 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.
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 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).
CYCLE_TIME_MKS	read only	Video capture cycle time. **VSource** 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);

    /**
     * @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.
     * @param mask Pointer to parameters mask.
     */
    void encode(uint8_t* data, int& size, VSourceParamsMask* mask = nullptr);
    /**
     * @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	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.

Field	type	Description
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. **VSource** 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:

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

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

Parameter	Value
size	Size of encoded data. 43 bytes by default.
mask	Parameters mask - pointer to **VSourceParamsMask** structure. **VSourceParamsMask** (declared in VSource.h file) determines flags for each field (parameter) declared in **VSourceParams** 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, 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, 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);
```

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:

```
// 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:

```
// 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:

```
{
    "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 respository 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)
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 (VSource class test applications) 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!