VSourceOpenCv C++ library

v1.1.1

Table of contents

- Overview
- Versions
- Library files
- VSourceOpenCv class description
  - VSourceOpenCv class declaration
  - getVersion method
  - openVSource method
  - initVSource method
  - isVSourceOpen method
  - closeVSource method
  - getFrame method
  - setParam method
  - getParam method
  - getParams method
  - executeCommand method
  - decodeAndExecuteCommand method
  - encodeSetParamCommand method of VSource interface class
  - encodeCommand method of VSource interface class
  - decodeCommand method of VSource interface class
- Data structures
  - VSourceCommand enum
  - VSourceParam enum
- VSourceParams class description
  - VSourceParams class declaration
  - Encode video source params
  - Decode video source params
  - Read params from JSON file and write to JSON file
- Build and connect to your project

- Simple example

Overview

VSourceOpenCv C++ library provides video capture and video source control function based on OpenCV library (version >=4.5). It provides simple interface for video capturing supported by OpenCV. The library inherits interface from open source VSource interface class. **VSource.h** file contains data structures VSourceParams class (contains video source params and provides methods for serialization/deserialization params), VSourceCommand enum (describes video source action commands), VSourceParam enum (describes video source params) and includes VSourceOpenCv class declaration. **VSourceOpenCv** depends on: OpenCV (version >=4.5), VSource interface class and open source Logger library which provides method to write logs. The library supports C++17 standard and provide simple interface.

Versions

Table 1 - Library versions.

Version	Release date	What's new
1.0.0	11.07.2023	First version.
1.0.1	13.11.2023	- VSource class updated. - Logger class updated.
1.0.2	13.11.2023	- CMake structure updated.
1.0.3	15.12.2023	- Auto reconnection issue fixed.
1.1.0	27.12.2023	- Region of interest support added.
1.1.1	09.01.2024	- Documentation updated.

Library files

The library is supplied only by source code. The user is given a set of files in the form of a CMake project (repository). The repository structure is shown below:

```
CMakeLists.txt ------------------- Main CMake file.
3rdparty ------------------------- Folder with third-party libraries.
    CMakeLists.txt --------------- CMake file to include third-party libraries.
    Logger ----------------------- Folder with files of Logger library.
    VSource ---------------------- Folder with files of VSource library.
src ------------------------------ Folder with library source code.
    CMakeLists.txt --------------- CMake file.
    VSourceOpenCv.h -------------- Main library header file.
    VSourceOpenCvVersion.h ------- Header file with library version.
    VSourceOpenCvVersion.h.in ---- File for CMake to generate version header.
    VSourceOpenCv.cpp ------------ C++ implementation file.
```

```
test ---------------------------- Folder for test application files.
    CMakeLists.txt --------------- CMake file for test application.
    main.cpp --------------------- Source C++ file of test application.
example ------------------------- Folder for example application.
    CMakeLists.txt --------------- CMake file for example application.
    main.cpp --------------------- Source C++ file of example application.
```

VSourceOpenCv class description

VSourceOpenCv class declaration

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

```
class VSourceOpenCv : public VSource
{
public:
    /// Class constructor.
    VSourceOpenCv();

    /// Class destructor.
    ~VSourceOpenCv();

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

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

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

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

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

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

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

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

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

    /// Execute command.
```

```
    bool executeCommand(VSourceCommand id) override;

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

getVersion method

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

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

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

```
cout << "VSourceOpenCv version: " << VSourceOpencv::getVersion() << endl;
```

Console output:

```
VSourceOpenCv version: 1.1.1
```

openVSource method

openVSource(...) method initializes video source. Video source params (VSourceParams class) will be initialized by default. Instead of openVSource(...) method user can call initVSource(...) method. Method declaration:

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

Parameter	Value
initString	Initialization string. Valid formats: **[source];[width];[height]** or **[source]** Initialization string parameters: **[source]** can be: file name (e.g. "test.mp4"), camera num (e.g. "0", "1" etc.) or rtsp stream (e.g. "rtsp://127.0.0.1:7031/live"). **[width]** optional frame width. Can be -1 for auto detection. **[height]** optional frame height. Can be -1 for auto detection. Example: "0;640;480" - open first camera in system with resolution 640x480.

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 class. The video source will set parameters according to params structure. If particular parameters not valid the library will set it to default.

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:

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

getFrame method

getFrame(...) method returns video frame. Video source supports auto reinitialization in case connection loss. In case capturing video from video file the method start playback again after end of file. Method declaration:

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

Parameter	Value
frame	Output video frame (see Frame class description). Frame should be initialized with **Fourcc::BGR24**, if not it will be automatically changed.

Parameter	Value
timeoutMsec	Timeout to wait new frame data, milliseconds: - 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 frame exists and copied or FALSE if not (wait timeout expired or video source not initialized).

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:

```
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 if the parameter was set or FALSE if not (not valid value or not supported).

getParam method

getParam(...) method returns 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 returns params structures. Method declaration:

```
void getParams(VSourceParams& params) override;
```

Parameter	Description
params	Video source parameters structure (parameters class).

executeCommand method

executeCommand(...) method executes video source command. Method declaration:

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

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

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

decodeAndExecuteCommand method

decodeAndExecuteCommand(...) method decodes and executes command on video source side. Method will decode commands which encoded by encodeCommand(...) and encodeSetParamCommand(...) methods of VSource interface class. If command decoded the method will call setParam(...) or executeCommand(...) methods. **decodeAndExecuteCommand(...)** is thread-safe. This means that the method can be safely called from any thread. Method declaration:

```
bool decodeAndExecuteCommand(uint8_t* data, int size) override;
```

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 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(...)** method encodes 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	Major	Major version of VSource class.
2	Minor	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:

```
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 VSourceCommand enum.

COMMAND format:

Byte	Value	Description
0	0x00	COMMAND header value.
1	Major	Major version of VSource class.
2	Minor	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 of VSource interface class

decodeCommand(...) static method of VSource interface class designed to decode command on video source side (edge device). Method will decode commands which encoded by encodeCommand(...) and encodeSetParamCommand(...) methods of VSource interface class. 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 7 bytes for COMMAND or 11 bytes for SET_PARAM command.
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 action commands (VSourceCommand enum).

VSourceCommand enum

Enum declaration:

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

Table 2 - Video source commands description.

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,
    /// Region of interest upper left corner x coordinate.
    ROI_X,
    /// Region of interest upper left corner y coordinate.
    ROI_Y,
    /// Region of interest width.
    ROI_WIDTH,
    /// Region of interest height.
    ROI_HEIGHT,
    /// [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 are not supported in **VSourceOpenCv** implementation.

Parameter	Access	Description
LOG_LEVEL	read / write	Logging mode. Specifies the log output mode. Values: 0 - Disable, 1 - Only file, 2 - Only terminal (console), 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	**Not supported.** Can have any value.
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. Valid values: 0 - Manual control, 1 - Auto.
EXPOSURE	read / write	Exposure value. Value: 0(min) - 65535(max).
FOCUS_MODE	read / write	Focus mode. Valid values: 0 - Manual control, 1 - Auto.
FOCUS_POS	read / write	Focus position. Value: 0(full near) - 65535(full far). Some cameras can support part of range 0-65535.
CYCLE_TIME_MKS	read only	Video capture cycle time, microseconds. **VSourceOpenCv** class sets this value automatically depending on input FPS value. 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.
ROI_X	read / write	Region of interest top-left corner horizontal coordinate.
ROI_Y	read / write	Region of interest top-left corner vertical coordinate.

Parameter	Access	Description
ROI_WIDTH	read / write	Region of interest width, pixels.
ROI_HEIGHT	read / write	Region of interest height, pixels.
CUSTOM_1	read / write	**Not supported.** Can have any value.**
CUSTOM_2	read / write	**Not supported.** Can have any value.
CUSTOM_3	read / write	**Not supported.** Can have any value.

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};
    /// Region of intrest upper left corner x coordinate.
    int roiX{0};
    /// Region of intrest upper left corner y coordinate.
    int roiY{0};
    /// Region of intrest width.
    int roiWidth{0};
    /// Region of intrest heigth.
    int roiHeight{0};
    /// 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, roiX, roiY,
                  roiWidth, roiHeight, custom1, custom2, custom3);

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

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

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

Table 4 - VSourceParams class fields description.

Field	type	Description
logLevel	int	Logging mode. Specifies the log output mode. Values: 0 - Disable, 1 - Only file, 2 - Only terminal (console), 3 - File and terminal.
source	string	Video source. Can be: file name (e.g. "test.mp4"), camera num (e.g. "0", "1" etc.) or rtsp stream (e.g. "rtsp://127.0.0.1:7031/live").
fourcc	string	Pixel format. Always will be BGR24.
width	int	Video frame width. If value **>0** the library will try set required resolution to video source if it not video file.
height	int	Video frame height. If value **>0** the library will try set required resolution to video source if it not video file.
gainMode	int	**Not supported.** Can have any value.
gain	int	Gain value. Value: 0(min for particular video source class) - 65535(max for particular video source class).
exposureMode	int	Exposure mode. Valid values: 0 - Manual control, 1 - Auto.
exposure	int	Exposure value. Value: 0(min) - 65535(max).
focusMode	int	Focus mode. Valid values: 0 - Manual control, 1 - Auto.
focusPos	int	Focus position. Value: 0(full near) - 65535(full far). Some cameras can support part of range 0-65535.
cycleTimeMks	int	Video capture cycle time, microseconds. **VSourceOpenCv** class sets this value automatically depending on input FPS value. This parameter means time interval between two captured video frame.
fps	float	FPS. Can't be changes after initialization.
isOpen	bool	Open flag. false - not open, true - open.
roiX	int	Region of interest top-left corner horizontal coordinate.
roiY	int	Region of interest top-left corner vertical coordinate.
roiWidth	int	Region of interest width, pixels.
roiHeight	int	Region of interest height, pixels.
custom1	float	**Not supported.** Can have any value.
custom2	float	**Not supported.** Can have any value.
custom3	float	**Not supported.** Can have any value.

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 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.
size	Size of encoded data. 78 bytes by default.
bufferSize	Data buffer size. If buffer size smaller than required, buffer will be filled with fewer parameters.
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:

```
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 roiX{true};
    bool roiY{true};
    bool roiWidth{true};
    bool roiHeight{true};
    bool custom1{true};
    bool custom2{true};
    bool custom3{true};
};
```

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;
```

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 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 **62** 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.
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": 150.0,
        "custom2": 252.0,
        "custom3": 30.0,
        "exposureMode": 226,
        "focusMode": 89,
        "fourcc": "skdfjhvk",
        "fps": 206.0,
        "gainMode": 180,
        "height": 61,
        "logLevel": 17,
        "roiHeight": 249,
        "roiWidth": 167,
        "roiX": 39,
        "roiY": 223,
        "source": "alsfghljb",
        "width": 35
```

```
    }
}
```

Build and connect to your project

Typical commands to build **VSourceOpenCv** library (OpenCV should be installed in your system):

```
cd VSourceOpenCv
git submodule update --init --recursive
mkdir build
cd build
cmake ..
make
```

If you want connect **VSourceOpenCv** 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
```

Create folder **3rdparty** in your repository. Copy repository folder **VSourceOpenCv** to **3rdparty** folder. New structure of your repository:

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

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(${PARENT}_VSOURCE_OPENCV                        ON  CACHE BOOL "" FORCE)
    SET(${PARENT}_VSOURCE_OPENCV_TEST                   OFF CACHE BOOL "" FORCE)
    SET(${PARENT}_VSOURCE_OPENCV_EXAMPLE                OFF CACHE BOOL "" FORCE)
endif()

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

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

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

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

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

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:

```
#include <iostream>
#include "VSourceOpenCv.h"
```

```
// Entry point.
int main(void)
{
    // Open camera with certain resolution.
    cr::video::VSource* source = new cr::video::VSourceOpenCv();
    if (source->openVSource("0;640;480"))
        return -1;

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

        // Prepare OpenCV frame to display.
        cv::Mat openCvFrame(frame.height, frame.width, CV_8UC3, frame.data);

        // Display frame.
        cv::imshow("VIDEO", openCvFrame);
        cv::waitKey(1);
    }

    return 1;
}
```