VSource interface C++ library

v1.0.0

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 form **VSource** C++ class to provide standard control interface. **VSource.h** file contains list of data structures (**VSourceParams** class, **VSourceCommand** enum and **VSourceParam** enum) and **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 method 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

Video source interface class description

Class declaration

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

```
namespace cr
{
namespace video
{
/**
 * @brief Video source interface class.
 */
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:
     * 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.
     * @return TRUE if new data exists 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 parameter value.
     * @param id Parameter ID according to camera specification.
     * @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;
};
}
}
```

getVersion method

getVersion() method return 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;
```

openVSource method

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

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

Parameter	Value
initString	Initialization string. Particular video source class can have specific format. Default format: [video device or ID or file];[width];[height];[fourcc].

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

initVSource method

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

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

Parameter	Value
params	VSourceParams structure (see **VSourceParams** description). The video source should set parameters according to params structure. Particular video source can support not all parameters listed in VSourceParams structure.

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

isVSourceOpen method

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

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

Returns: TRUE if the video source open 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. 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.

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.
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: parameter video source parameters structure.

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.

Data structures

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

VSourceCommand enum

Enum declaration:

```
enum class VSourceCommand
{
    /// Restart.
    RESTART = 1,
    /// Apply settings.
    APPLY_PARAMS
};
```

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).
APPLY_PARAMS	Apply (Save) parameters.

VSourceParam enum

Enum declaration:

```
enum class VSourceParam
{
    /// [read/write] Log level:
    /// 0-Disable, 1-Console, 2-File, 3-Console and file.
    LOG_LEVEL = 1,
    /// [read/write] Frame width.
    WIDTH,
    /// [read/write] Frame height.
    HEIGHT,
    /// [read/write] Gain mode. Depends on implementation.
    /// Default: 0 - Manual, 1 - Auto.
    GAIN_MODE,
    /// [read/write] Gain value in case manual gain mode.
    /// Value: 0(min) - 65535(max).
    GAIN,
    /// [read/write] Exposure mode. Depends on implementation.
    /// Default: 0 - Manual, 1 - Auto.
    EXPOSURE_MODE,
    /// [read/write] Exposure value in case manual exposure mode.
    /// Value: 0(min) - 65535(max).
    EXPOSURE,
    /// [read/write] Focus mode. Depends on implementation.
    /// Default: 0 - Manual, 1 - Auto.
    FOCUS_MODE,
    /// [read/write] Focus position.
    /// Value: 0(full near) - 65535(full far).
    FOCUS_POS,
    /// [read only] Cycle processing time microsecconds.
    CYCLE_TIME_MKS,
    /// [read/write]  FPS. 0 - will be set automatically.
    FPS,
    /// Open flag. 0 - not open, 1 - open.
    IS_OPEN
};
```

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

Parameter	Access	Description
LOG_LEVEL	read / write	Logging mode. Default 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.

Parameter	Access	Description
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. 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. 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. 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.
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.

VSourceParams class

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) and provide methos to encode and decode params. Class declaration:

```
class VSourceParams
{
public:
    /// Log level: Disable, Console, File, Console and file.
    std::string logLevel{"Disable"};
    /// Video source name.
    std::string source{"/dev/video0"};
    /// FOURCC: RGB24, BGR24, YUYV, UYVY, GRAY, YUV24, NV12, NV21, YU12, YV12.
    std::string fourcc{"YUYV"};
    /// Frame width. 0 - will be set automatically.
    int width{1920};
    /// Frame height. 0 - will be set automatically.
    int height{1080};
    /// Gain mode. Depends on implementation. Default: 0 - Manual, 1 - Auto.
    int gainMode{1};
```

```
    /// Gain value in case manual gain mode. Value: 0(min) - 65535(max).
    int gain{0};
    /// Exposure mode. Depends on implementation. Default: 0 - Manual, 1 - Auto.
    int exposureMode{1};
    /// Exposure value in case manual exposure mode. Value: 0(min) - 65535(max).
    int exposure{1};
    /// Focus mode. Depends on implementation. Default: 0 - Manual, 1 - Auto.
    int focusMode{1};
    /// Focus position. Value: 0(full near) - 65535(full far).
    int focusPos{0};
    /// Cycle processing time microsecconds.
    int cycleTimeMks{0};
    /// FPS. 0 - will be set automatically.
    float fps{0};
    /// Open flag.
    bool isOpen{false};

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

    /**
     * @brief operator =
     * @param src Source object.
     * @return VSourceParams obect.
     */
    VSourceParams& operator= (const VSourceParams& src);

    /**
     * @brief Encode params. The method doesn't encode params:
     * logLevel, source and fourcc.
     * @param data Pointer to data buffer.
     * @param size Size of data.
     */
    void encode(uint8_t* data, int& size);

    /**
     * @brief Decode params. The method doesn't decode params:
     * logLevel, source and fourcc.
     * @param data Pointer to data.
     * @return TRUE is params decoded or FALSE if not.
     */
    bool decode(uint8_t* data, int size);
};
```

Table 4 - VSourceParams class fields description.

Field	type	Description
logLevel	string	Logging mode. Default values: "Disable" - Disable printing log info. "File" - Only printing in file. "Console" - Only printing in terminal. "Console and file" - File and terminal printing.
source	string	Video device name (depends on OS), video source or file. Format depends on particular video source class.
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. 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. 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. 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.
fps	float	FPS. User can set frame FPS before initialization or after. Some video source classes may set FPS automatically.
isOpen	bool	Open flag. FLASE - video source not open, TRUE - video source open.

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

Encode (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: **logLevel**, **source** and **fourcc**. Method declaration:

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

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

Example:

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

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

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

Decode (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: **logLevel**, **source** and **fourcc**. Method declaration:

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

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

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, size))
    cout << "Can't decode data" << endl;
```