VStreamer interface C++ library

v1.1.1

Table of contents

Overview

VStreamer C++ library provides standard interface as well defines data structures and rules for different video stream classes. **VStreamer** interface class doesn't do anything, just provides interface and methods to encode/decode commands and encode/decode params. Also **VStreamer** class provides data structures for video stream parameters. Different video stream classes inherit interface form **VStreamer** C++ class. **VStreamer.h** file contains list of data structures (VStreamCommand enum, VStreamParam enum and VStreamParams class). VStreamParams class contains video stream params and includes methods to encode and decode params. VStreamCommand enum contains IDs of commands supported by **VStreamer** class. VStreamParam enum contains IDs of params supported by **VStreamer** class. All video streamers should include params and commands listed in **VStreamer.h** file. **VStreamer** class interface class depends on: ConfigReader library provides methods to read/write JSON config files, VCodec interface library for integrating codec implementations in case raw frame streaming, VOverlay interface library for integrating overlay engines in case raw frame streaming. The library is licensed under the **Apache 2.0** license.

Versions

Table 1 - Library versions.

Version	Release date	What's new
1.0.0	15.02.2024	First version of the library.
1.1.0	26.02.2024	- New parameters added for multicast streaming.
1.1.1	20.03.2024	- VCodec class updated. - VOverlay class updated. - Frame subrepository excluded. - Documentation updated.

Library files

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

```
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.
    VCodec ----------------------- Source code of the VCodec library.
    VOverlay --------------------- Source code of the VOverlay library.
example -------------------------- Folder with simple example of VStreamer
implementation.
    CMakeLists.txt --------------- CMake file for example custom video streamer class.
    CustomVStreamer.cpp ---------- Source code file of the CustomVStreamer class.
    CustomVStreamer.h ------------ Header with CustomVStreamer class declaration.
```

```
    CustomVStreamerVersion.h ----- Header file which includes CustomVStreamer version.
    CustomVStreamerVersion.h.in -- CMake service file to generate version file.
test ----------------------------- Folder with VStreamer test application.
    CMakeLists.txt --------------- CMake file for VStreamer test application.
    main.cpp --------------------- Source code file of VStreamer class test application.
src ------------------------------ Folder with source code of the library.
    CMakeLists.txt --------------- CMake file of the library.
    VStreamer.cpp ---------------- Source code file of the library.
    VStreamer.h ------------------ Header file which includes VStreamer class
declaration.
    VStreamerVersion.h ----------- Header file which includes version of the library.
    VStreamerVersion.h.in -------- CMake service file to generate version file.
```

Video streamer interface class description

VStreamer class declaration

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

```
class VStreamer
{
public:

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

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

    /// Init video streamer.
    virtual bool initVStreamer(VStreamerParams &params,
                               VCodec *codec = nullptr,
                               VOverlay *overlay = nullptr) = 0;

    /// Get initialization status.
    virtual bool isVStreamerInit() = 0;

    /// Close video streamer.
    virtual void closeVStreamer() = 0;

    /// Send frame to video streamer.
    virtual bool sendFrame(Frame& frame) = 0;

    /// Set video streamer param with int type value.
    virtual bool setParam(VStreamerParam id, float value) = 0;

    /// Set video streamer param with string type value.
    virtual bool setParam(VStreamerParam id, std::string value) = 0;

    /// Get video streamer params structure.
    virtual void getParams(VStreamerParams& params) = 0;
```

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

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

    /// Encode set param command.
    static void encodeSetParamCommand(
            uint8_t* data, int& size, VStreamerParam id,
            float value1, std::string value2 = "");

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

    /// Decode command.
    static int decodeCommand(uint8_t* data,
                             int size,
                             VStreamerParam& paramId,
                             VStreamerCommand& commandId,
                             float& value1,
                             std::string& value1);
};
```

getVersion method

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

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

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

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

Console output:

```
VStreamer class version: 1.1.1
```

initVStreamer method

The **initVStreamer(...)** method initializes video streamer by set of parameters. Method declaration:

```
virtual bool initVStreamer(VStreamerParams &params,
                           VCodec *codec = nullptr,
                           VOverlay *overlay = nullptr) = 0;
```

Parameter	Value
params	VStreamerParams class object. The video streamer should set parameters according to params structure. Particular video streamer might not support all parameters listed in VStreamerParams class.
codec	Pinter VCodec object. Used for encoding video in case RAW input frame data. If user set pointer to **nullptr** the video streamer can process only compressed input video frames.
overlay	Pointer to VOverlay object. Used to overlay information on video in case if user put RAW input frame data to the streamer. If user set pointer to **nullptr** the video streamer will not be able overlay any information on video.

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

isVStreamerInit method

The **isVStreamerInit()** method returns video streamer initialization status. Method declaration:

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

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

closeVStreamer method

The **closeVStreamer()** method closes video streamer. Method stops all thread and releases memory. Method declaration:

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

sendFrame method

The **sendFrame(...)** method intended to send frame to clients. To provide video stream the user must call this method for every video frame coming from video source. In case processing RAW video frames streamer must rescale video (if necessary) and overlay information (if pointer to video overlay class set in initVStreamer(...) method). Method declaration:

```
virtual bool sendFrame(Frame& frame) = 0;
```

Parameter	Value
frame	Frame class object. Particular streamers can support RAW video frames or(and) compressed video frames.

Returns: TRUE if frame sent (accepted by streamer) or FALSE if not.

setParam method

The **setParam(...)** method sets new value of video stream parameter. The particular implementation of the video streamer must provide thread-safe **setParam(...)** method call. This means that the **setParam(...)** method can be safely called from any thread. Also, method has two overloaded version that depends on type of value. Method declaration:

```
virtual bool setParam(VStreamerParam id, float value) = 0;
virtual bool setParam(VStreamerParam id, std::string value) = 0;
```

Parameter	Description
id	Video stream parameter ID according to VStreamParam enum.
value	Value of parameter. It can be either string or int type. it depends on parameter.

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

getParams method

The **getParams(...)** method to obtain video streamer params class. The particular implementation of the video streamer 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(VStreamerParams& params) = 0;
```

Parameter	Description
params	VStreamerParams class object.

executeCommand method

The **executeCommand(...)** method to execute video stream command. The particular implementation of the video stream 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(VStreamerCommand id) = 0;
```

Parameter	Description
id	Video stream command ID according to VStreamCommand enum.

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

decodeAndExecuteCommand method

The **decodeAndExecuteCommand(...)** method decodes and executes command encoded by encodeSetParamCommand(...) and encodeCommand(...) methods on video streamer side. It is a virtual method which means if implementation does not define it, default definition from **VStreamer** class will be used. The particular implementation of the video streamer must provide thread-safe **setParam(...)** and **executeCommand(...)** method calls to make default definition of **decodeAndExecuteCommand(...)** thread-safe. This means that the **decodeAndExecuteCommand(...)** method can be safely called from any thread. Method declaration:

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

Parameter	Description
data	Pointer to input command.
size	Size of command.

Returns: TRUE if command decoded (SET_PARAM or COMMAND) and executed (action command or set param command).

encodeSetParamCommand method

The **encodeSetParamCommand(...)** static method to encode command to change any parameter for remote video streamer. To control video streamer 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 **VStreamer** class contains static methods for encoding the control command. The **VStreamer** 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, VStreamerParam id,
            float value1, std::string value2 = "");
```

Parameter	Description
data	Pointer to data buffer for encoded command. Must have size >= 11.
size	Size of encoded data. Will be minimum 11 bytes.
id	Parameter ID according to VStreamerParam enum.
value1	Numeral video streamer parameter value. Only for non string parameters. For string parameters (see VStreamParam enum) this parameters may have any values.
value2	String parameter value (see VStreamParam enum).

encodeSetParamCommand(...) is static and used without **VStreamer** 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.
VStreamer::encodeSetParamCommand(data, size, VStreamerParam::CUSTOM1, outValue);
```

encodeCommand method

The **encodeCommand(...)** static method to encode command for remote video streamer. To control a video streamer 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 **VStreamer** class contains static methods for encoding the control command. The **VStreamer** 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, VStreamerCommand 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 VStreamerCommand enum.

encodeCommand(...) is static and used without **VStreamer** 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.
VStreamer::encodeCommand(data, size, VStreamerCommand::RESTART);
```

decodeCommand method

The **decodeCommand(...)** static method to decode command on video streamer side (edge device). Method declaration:

```
static int decodeCommand(uint8_t* data,
                         int size,
                         VStreamerParam& paramId,
                         VStreamerCommand& commandId,
                         float& value1,
                         std::string& value1);
```

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 VStreamerParam enum. After decoding SET_PARAM command the method will return parameter ID.
commandId	Command ID according to VStreamerCommand enum. After decoding COMMAND the method will return command ID.
value1	Numeral video streamer parameter value. Only for non string parameters. For string parameters (see VStreamParam enum) this parameters may have any values.
value2	String parameter value (see VStreamParam enum).

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

Data structures

VStreamer.h file defines IDs for parameters (**VStreamerParam** enum) and IDs for commands (**VStreamerCommand** enum).

VStreamerCommand enum

Enum declaration:

```
enum class VStreamerCommand
{
    /// Restart.
    RESTART = 1,
    /// Enable. Equal to MODE param.
    ON,
    /// Disable. Equal to MODE params.
    OFF
};
```

Table 4 - Video stream commands description. Some commands maybe unsupported by particular video stream class.

Command	Description
RESTART	Restarts streamer with last VStreamerParams.
ON	Enables streamer if it is disabled.
OFF	Disables streamer if it is enabled.

VStreamerParam enum

Enum declaration:

```
enum class VStreamerParam
{
    /// Mode: 0 - disabled, 1 - enabled.
    MODE = 1,
    /// Video stream width from 8 to 8192.
    WIDTH,
    /// Video stream height from 8 to 8192.
    HEIGHT,
    /// Streamer IP.
    IP,
    /// Streamer port.
    PORT,
    /// Streamer multicast IP.
    MULTICAST_IP,
    /// Streamer multicast port.
    MULTICAST_PORT,
    /// Streamer user (for rtsp streaming): "" - no user.
    USER,
    /// Streamer password (for rtsp streaming): "" - no password.
    PASSWORD,
    /// Streamer suffix(for rtsp streaming, stream name).
    SUFFIX,
    /// Minimum bitrate for variable bitrate mode, kbps.
    MIN_BITRATE_KBPS,
    /// Maximum bitrate for variable bitrate mode, kbps.
    MAX_BITRATE_KBPS,
    /// Current bitrate, kbps.
    BITRATE_KBPS,
    /// Bitrate mode: 0 - constant bitrate, 1 - variable bitrate.
    BITRATE_MODE,
    /// FPS.
    FPS,
    /// GOP size for H264 and H265 codecs.
    GOP,
    /// H264 profile: 0 - baseline, 1 - main, 2 - high.
    H264_PROFILE,
    /// JPEG quality from 1 to 100% for JPEG codec.
    JPEG_QUALITY,
    /// Codec type: "H264", "HEVC" or "JPEG".
    CODEC,
    /// Scaling mode: 0 - fit, 1 - cut.
```

```
    FIT_MODE,
    /// Overlay mode: false - off, true - on.
    OVERLAY_MODE,
    /// TYPE of the streamer
    TYPE,
    /// Custom parameter 1.
    CUSTOM1,
    /// Custom parameter 2.
    CUSTOM2,
    /// Custom parameter 3.
    CUSTOM3
};
```

Table 5 - Video streamer params description. Some params maybe unsupported by particular video streamer class.

Parameter	Description
MODE	Enable/disable streamer (0 - disabled, 1 - enabled).
WIDTH	Frame width. Regardless of the resolution of the input video, if RAW data is processed, the streamer should scale the images according to this parameter.
HEIGHT	Frame height. Regardless of the resolution of the input video, if RAW data is processed, the streamer should scale the images according to this parameter.
IP	Streamer's ip.
PORT	Streamer's port. It can be TCP or UDP port depends on implementation.
MULTICAST_IP	Streamer's multicast ip.
MULTICAST_PORT	Streamer's multicast port. It is UDP port.
USER	User name for auth (for example, in case RTSP stream).
PASSWORD	Password name for auth (for example, in case RTSP stream).
SUFFIX	Stream name in case RTSP stream.
MIN_BITRATE_KBPS	Minimum bitrate for variable bitrate encoding in case RAW input video frames.
MAX_BITRATE_KBPS	Maximum bitrate for variable bitrate encoding in case RAW input video frames.
BITRATE_KBPS	Bitrate for constant bitrate encoding in case RAW input video frames.
BITRATE_MODE	Enable/disable variable bitrate. Values depends on implementation but it is recommended to use default values: **0** - constant bitrate, **1** - variable bitrate.
FPS	Streamer's fps and also encoding fps in case RAW input video frames. Regardless of the input video frame rate, the streamer must provide the specified FPS. If the FPS value is 0, the streamer must provide FPS equal to the input video frame rate.
GOP	Codec gop size in case in case RAW input video frames.
H264_PROFILE	H264 encoding profile in case RAW input video frames.

Parameter	Description
JPEG_QUALITY	JPEG encoding quality in case RAW input video frames.
CODEC	Codec type for encoding RAW frames.
FIT_MODE	Scaling mode. Values depends on implementation but it is recommended to use default values: **0** - fit, **1** - crop. in case RAW input video frames.
OVERLAY_MODE	Overlay enable/disable in case RAW input video frames. Values depends on implementation but it is recommended to use default values: **0** - disable, **1** - enable.
CYCLE_TIME_MKSEC	**Read only.** Cycle timeout, microseconds.
TYPE	Type of streamer. Depends on implementation.
CUSTOM1	Custom parameter. Depends on implementation.
CUSTOM2	Custom parameter. Depends on implementation.
CUSTOM3	Custom parameter. Depends on implementation.

VStreamerParams class description

VStreamerParams class declaration

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

```
class VStreamerParams
{
public:
    /// Streamer mode: false - Off, true - On.
    bool enable{true};
    /// Video stream width from 8 to 8192.
    int width{1280};
    /// Video stream height from 8 to 8192.
    int height{720};
    /// Streamer IP.
    std::string ip{"127.0.0.1"};
    /// Streamer port.
    int port{8554};
    /// Streamer multicast IP.
    std::string multicastIp{"224.1.0.1"};
    /// Streamer multicast port.
    int multicastPort{18000};
    /// Streamer user (for rtsp streaming): "" - no user.
    std::string user{""};
```

```
    /// Streamer password (for rtsp streaming): "" - no password.
    std::string password{""};
    /// Streamer suffix (for rtsp streaming) (stream name).
    std::string suffix{"live"};
    /// Minimum bitrate for variable bitrate mode, kbps.
    int minBitrateKbps{1000};
    /// Maximum bitrate for variable bitrate mode, kbps.
    int maxBitrateKbps{5000};
    /// Current bitrate, kbps.
    int bitrateKbps{3000};
    /// Bitrate mode: 0 - constant bitrate, 1 - variable bitrate.
    int bitrateMode{0};
    /// FPS.
    float fps{30.0f};
    /// GOP size for H264 and H265 codecs.
    int gop{30};
    /// H264 profile: 0 - baseline, 1 - main, 2 - high.
    int h264Profile{0};
    /// JPEG quality from 1 to 100% for JPEG codec.
    int jpegQuality{80};
    /// Codec type: "H264", "HEVC" or "JPEG".
    std::string codec{"H264"};
    /// Scaling mode: 0 - fit, 1 - cut.
    int fitMode{0};
    /// Cycle time, mksec. Calculated by streamer.
    int cycleTimeMksec{0};
    /// Overlay mode: false - off, true - on.
    bool overlayMode{true};
    /// type of the streamer
    int type{0};
    /// Custom parameter 1.
    float custom1{0.0f};
    /// Custom parameter 2.
    float custom2{0.0f};
    /// Custom parameter 3.
    float custom3{0.0f};

    JSON_READABLE(VStreamerParams, enable, width, height, ip, port, user,
                  password, suffix, minBitrateKbps, maxBitrateKbps, bitrateKbps,
                  bitrateMode, fps, gop, h264Profile, jpegQuality, codec,
                  fitMode, overlayMode, type, custom1, custom2, custom3)

    /// Assignment operator.
    VStreamerParams& operator= (const VStreamerParams& src);

    /// Serialize parameters.
    bool encode(uint8_t* data, int bufferSize, int& size,
                VStreamerParamsMask* mask = nullptr);

    /// Deserialize parameters.
    bool decode(uint8_t* data, int dataSize);
};
```

Serialize video streamer params

VStreamerParams class provides method **encode(...)** to serialize video streamer params (fields of VStreamerParams class). Serialization of video streamer params necessary in case when you need to send video streamer params via communication channels. Method provides options to exclude particular parameters from serialization. To do this method inserts binary mask (4 bytes) where each bit represents particular parameter and **decode(...)** method recognizes it. Method declaration:

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

Parameter	Value
data	Pointer to data buffer.
size	Size of encoded data.
bufferSize	Data buffer size. If buffer size smaller than required, buffer will be filled with fewer parameters.
mask	Parameters mask - pointer to **VStreamerParamsMask** structure. **VStreamerParamsMask** (declared in VStreamer.h file) determines flags for each field (parameter) declared in **VStreamerParams** 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 VStreamerParams structure.

VStreamerParamsMask structure declaration:

```
struct VStreamerParamsMask
{
    bool enable{true};
    bool width{true};
    bool height{true};
    bool ip{true};
    bool port{true};
    bool multicastIp{true};
    bool multicastPort{true};
    bool user{true};
    bool password{true};
    bool suffix{true};
    bool minBitrateKbps{true};
    bool maxBitrateKbps{true};
    bool bitrateKbps{true};
    bool bitrateMode{true};
    bool fps{true};
    bool gop{true};
    bool h264Profile{true};
    bool jpegQuality{true};
    bool codec{true};
    bool fitMode{true};
    bool cycleTimeMksec{true};
    bool overlayMode{true};
```

```
    bool type{true};
    bool custom1{true};
    bool custom2{true};
    bool custom3{true};
};
```

Example without parameters mask:

```
// Prepare random params.
VStreamerParams in;
in.ip = "alsfghljb";
in.port = 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.
VStreamerParams in;
in.ip = "alsfghljb";
in.port = 0;

// Prepare params mask.
VStreamerParamsMask mask;
mask.port = false; // Exclude port. 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 stream params

VStreamerParams class provides method **decode(...)** to deserialize video streamer params (fields of **VStreamerParams** class). Deserialization of video streamer params necessary in case when you need to receive video streamer params via communication channels. 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.
dataSize	Size of data.

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

Example:

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

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

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

Read params from JSON file and write to JSON file

VStreamer 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.
VStreamerParams in;
cr::utils::ConfigReader inConfig;
inConfig.set(in, "vStreamerParams");
inConfig.writeToFile("TestVStreamerParams.json");

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

TestVStreamerParams.json will look like:

```
{
    "vStreamerParams":
    {
        "bitrateKbps": 207,
        "bitrateMode": 206,
        "codec": "eydiucnksa",
        "custom1": 249.0,
        "custom2": 150.0,
        "custom3": 252.0,
        "enable": false,
        "fitMode": 39,
        "fps": 35.0,
        "gop": 1,
        "h264Profile": 61,
```

```
        "height": 61,
        "ip": "afhjaskdm",
        "jpegQuality": 80,
        "maxBitrateKbps": 89,
        "minBitrateKbps": 180,
        "multicastIp": "afhjaskdmasd",
        "multicastPort": 180,
        "overlayMode": true,
        "password": "adafsodjf",
        "port": 226,
        "suffix": "asdasdasd",
        "type": 167,
        "user": "afhidsjfnm",
        "width": 50
    }
}
```

Build and connect to your project

Typical commands to build **VStreamer** library:

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

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

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

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

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

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(${PARENT}_SUBMODULE_VSTREAMER                       ON  CACHE BOOL "" FORCE)
if (${PARENT}_SUBMODULE_VSTREAMER)
    SET(${PARENT}_VSTREAMER                             ON  CACHE BOOL "" FORCE)
    SET(${PARENT}_VSTREAMER_TEST                        OFF CACHE BOOL "" FORCE)
    SET(${PARENT}_VSTREAMER_EXAMPLE                     OFF CACHE BOOL "" FORCE)
endif()

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

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

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

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

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

Done!

How to make custom implementation

The **VStreamer** class provides only an interface, data structures, and methods for encoding and decoding commands and params. To create your own implementation of the video streamer, you must include the VStreamer 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 streamer. You must implement all the methods of the VStreamer interface class. Custom video streamer class declaration:

```
/// Custom video streamer class.
class CustomVStreamer: public VStreamer
{
public:

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

    /// Init video streamer.
    bool initVStreamer(VStreamerParams &params,
                       VCodec *codec = nullptr,
                       VOverlay *overlay = nullptr);

    /// Get initialization status.
    bool isVStreamerInit();

    /// Close video streamer.
    void closeVStreamer();

    /// Send frame to video streamer.
    bool sendFrame(Frame& frame)

    /// Set video streamer param with int value.
```

```
    bool setParam(VStreamerParam id, float value);

    /// Set video streamer param with string value.
    bool setParam(VStreamerParam id, string value);

    /// Get video streamer params structure.
    void getParams(VStreamerParams& params);

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

private:

    /// Video streamer params.
    VStreamerParams m_params;
};
```