RtspServer C++ library

v2.0.0

Table of contents

Overview

The **RtspServer** is a C++ library implements RTSP server. The **RtspServer** library inherits interface from VStreamer class, and can be included in any C++ project for Linux and Windows. It allows developer to create a RTSP server and set general parameters: RTSP server port, IP, stream name, user name and password. The library accepts RAW pixel formats (BGR24, RGB24, GRAY, YUYV, UYVY, NV12, NV21, YU12 and YV12) and compressed data (H264 and HEVC). It supports **H264** and **HEVC** video streams and also supports multicast video streaming. If the input video frame is in a compressed format (H264 or HEVC), it bypasses any overlay and resizing processes and is sent directly to clients. However, if the frame is RAW format, it undergoes a series of processing steps. First, resizing is performed if necessary, followed by the application of any required overlay. Finally, the processed frame is encoded by video codec before being transmitted to clients (to use this functions the user must define video codec and video overlay implementation according to VCodec, VOverlay interfaces). The library relies on several dependencies, including VStreamer class (defines interface, source code included, Apache 2.0 license), **FormatConverterOpenCv** (provides methods to convert pixel formats, source code included) and OpenCV (version 4.5 and higher, Apache 2.0 license). Additionally example depends on **VSourceFile** library (provides interface to capture compressed video from files, source code included). It has been specifically tested and verified for functionality on the **Linux** and **Windows** operating systems. The RTSP server provided by library compatible with all popular RTSP clients: **ffmpeg**, **gstreamer**, **VLC**, **Milestone** etc.

Versions

Table 1 - Library versions.

Version	Release date	What's new
1.0.0	20.11.2023	- First version
2.0.0	27.02.2024	- Interface is changed. - Streaming Raw frame support added. - Multicast stream support added.

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 of the library.
3rdparty ----------------------- Folder with 3rdparty libraries.
    CMakeLists.txt ------------- CMake file to include 3rdparty libraries.
    FormatConverterOpenCv ------ Folder with FormatConverterOpenCv library source code.
    VStreamer ------------------ Folder with VStreamer interface library.
src ---------------------------- Folder with library source code.
    CMakeLists.txt ------------- CMake file.
    RtspServer.cpp ------------- Library main class source file.
    RtspServer.h --------------- Library main class header file.
```

```
    RtspServerVersion.h -------- Header file with library version.
    RtspServerVersion.h.in ----- Service CMake file to generate version file.
    RtspServerBackend ---------- Folder with backend library source code.
examples ----------------------- Folder with examples.
    CMakeLists.txt ------------- CMake file of test application.
    CompressedFrameStreaming --- Folder with example application.
        3rdparty --------------- Folder with 3rdpary libraries for example application.
            CMakeLists.txt ----- CMake file to include 3rdparty libraries.
            VSourceFile -------- Folder with VSourceFile library source code.
        CMakeLists.txt --------- CMake file of example application.
        main.cpp --------------- Source code of example application.
```

RtspServer class description

Class declaration

RtspServer class declared in **RtspServer.h** file. Class declaration:

```
class RtspServer : public cr::video::VStreamer
{
public:

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

    /// Class constructor.
    RtspServer();

    /// Class destructor.
    ~RtspServer();

    /// Init RTSP server.
    bool initVStreamer(cr::video::VStreamerParams &params,
                       cr::video::VCodec *codec = nullptr,
                       cr::video::VOverlay *overlay = nullptr) override;

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

    /// Close RTSP server.
    void closeVStreamer() override;

    /// Send frame to RTSP server.
    bool sendFrame(cr::video::Frame& frame) override;

    /// Set RTSP server parameter.
    bool setParam(cr::video::VStreamerParam id, float value) override;

    /// Set RTSP server parameter.
    bool setParam(cr::video::VStreamerParam id, std::string value) override;
```

```
    /// Get RTSP server parameters.
    void getParams(cr::video::VStreamerParams& params) override;

    /// Execute action command.
    bool executeCommand(cr::video::VStreamerCommand id) override;
};
```

getVersion method

The **getVersion()** method return string of current class version. Method declaration:

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

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

```
cout << "RtspServer version : " << RtspServer::getVersion() << endl;
```

Console output:

```
RtspServer version : 2.0.0
```

initVStreamer method

The **initVStreamer(...)** method initializes RTSP server. If server already initialized the method will reinitialize server. If RTSP server parameters changed user does not need to call **initVStreamer(...)** method to reinitialize server. Method declaration:

```
bool initVStreamer(VStreamerParams& params,
                   cr::video::VCodec* codec = nullptr,
                   cr::video::VOverlay* overlay = nullptr) override;
```

Parameter	Value
params	VStreamerParams object which includes initialization parameters.
codec	VCodec object pointer in case RAW input frames.
overlay	VOverlay object pointer to provide video overlay in case RAW input frames.

Important note: The VCodec implementation must be designed to accept input frames in the NV12 format. The RtspServer library feeds NV12 formatted frames into the codec. If a custom implementation of VCodec does not support NV12 as the input format, the library will not work properly. Similarly, the VOverlay implementation must also be designed to handle input frames, but in the YUV24 format instead of NV12. This requirement is essential for the correct operation of the library.

Returns: TRUE if initialization is done or FALSE if not.

isVStreamerInit method

isVStreamerInit() method returns RTSP server initialization status. Method declaration:

```
bool isVStreamerInit() override;
```

Returns: TRUE if the RTSP server initialized or FALSE if not.

closeVStreamer method

The **closeVStreamer()** method closes RTSP server if open. Method declaration:

```
void closeVStreamer() override;
```

sendFrame method

The **sendFrame(...)** method to send Frame to RTSP server. The method does not block the calling thread. The library has an internal thread of execution, which is passed the video frame for processing. Method declaration:

```
bool sendFrame(cr::video::Frame& frame) override;
```

Parameter	Value
frame	Frame object to send RTSP stream. H264, HEVC and raw frame formats are supported. In case of raw frame, VCodec object had to be provided to initVStreamer(...) method. Also in case of raw frame format, fitting-scaling will be carried out by server if provided frame has different dimensions than dimensions defined by RtspServerParams. If provided frame is not RAW, Overlay can not apply even if user provides VOverlay instance to initVStreamer method.

Returns: TRUE if frame is sent or FALSE if not.

setParam (float parameter) method

The **setParam(...)** method to set parameter with float value. This method is thread-safe. This means that the **setParam(...)** method can be safely called from any thread. Method declaration:

```
bool setParam(cr::video::VStreamerParam id, float value) override;
```

Parameter	Value
id	Parameter ID according to VStreamerParam enum class.
value	Float type parameter value.

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

setParam (string parameter) method

The **setParam(...)** method sets parameter with string value. This method is thread-safe. This means that the **setParam(...)** method can be safely called from any thread. Method declaration:

```
bool setParam(cr::video::VStreamerParam id, std::string value) override;
```

Parameter	Value
id	Parameter ID according to VStreamerParam enum class.
value	String type parameter value.

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

getParams method

The **getParams(...)** method to get current parameters that are defined in form of VStreamerParams class. This method is thread-safe. This means that the **getParams(...)** method can be safely called from any thread. Method declaration:

```
void getParams(cr::video::VStreamerParams& params) override;
```

Parameter	Value
params	Reference to VStreamerParams object to store current parameters.

executeCommand method

The **executeCommand(...)** method to execute RTSP server action command. This method is thread-safe. This means that the **executeCommand(...)** method can be safely called from any thread. Method declaration:

```
bool executeCommand(cr::video::VStreamerCommand id) override;
```

Parameter	Value
id	Command id according to VStreamerCommand enum.

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

decodeAndExecuteCommand method of VStreamer class

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. Must be min 11 bytes for SET_PARAM and 7 bytes for COMMAND.

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

encodeSetParamCommand method of VStreamer class

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 of VStreamer class

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 of VStreamer class

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.

Command	Description
RESTART	Restarts RTSP with last VStreamerParams.

Command	Description
ON	Enable RTSP server (resume).
OFF	Disable RTSP server (pause).

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 (0 - unicast, 1 - multicast)
    TYPE,
    /// Custom parameter 1.
    CUSTOM1,
    /// Custom parameter 2.
    CUSTOM2,
    /// Custom parameter 3.
    CUSTOM3
};
```

Table 5 - Video streamer params description.

Parameter	Description
MODE	Enable/disable RTSP server (0 - disabled, 1 - enabled).
WIDTH	Frame width. In case processing RAW frames (pixel formats BGR24, RGB24, GRAY, YUYV, UYVY, NV12, NV21, YU12 and YV12) the library will resize input image according to this parameter value.
HEIGHT	Frame height. In case processing RAW frames (pixel formats BGR24, RGB24, GRAY, YUYV, UYVY, NV12, NV21, YU12 and YV12) the library will resize input image according to this parameter value.
IP	RTSP server's ip. If the user doesn't know host IP he can set IP to **"0.0.0.0"**. It will provide default IP initialization.
PORT	RTSP server's port from 0 to 65535.
MULTICAST_IP	RTSP server's multicast ip. The library supports multicast IP streaming (UDP multicast). The library doesn't support requests to set particular multicast IP and port from client. If client requests particular multicast IP and port the user should set the same values in RTSP server parameters in advance.
MULTICAST_PORT	RTSP server's multicast UDP port. The library doesn't support requests to set particular multicast IP and port from client. If client requests particular multicast IP and port the user should set the same values in RTSP server parameters in advance.
USER	User name for auth of RTSP stream. If set to **""** the RTSP will work without auth.
PASSWORD	Password name for auth of RTSP stream. If set to **""** the RTSP will work without auth.
SUFFIX	Stream name for RTSP stream. For example "live".
MIN_BITRATE_KBPS	Minimum bitrate for variable bitrate encoding in case RAW input video frames. Not supported by **RtspServer** class.
MAX_BITRATE_KBPS	Maximum bitrate for variable bitrate encoding in case RAW input video frames. Not supported by **RtspServer** class.

Parameter	Description
BITRATE_KBPS	Constant bitrate encoding in case RAW input video frames (pixel formats BGR24, RGB24, GRAY, YUYV, UYVY, NV12, NV21, YU12 and YV12). The library will configure video codec paraneters if it provided by use (if user set pointer to VCodec class object).
BITRATE_MODE	Enable/disable variable bitrate. Not supported by **RtspServer** class.
FPS	Streamer's fps and also encoding fps in case RAW input video frames. Regardless of the input video frame rate, the streamer provides the specified FPS. If the FPS value is 0, the streamer provides FPS equal to the input video frame rate.
GOP	Codec gop size in case in case RAW input video frames (pixel formats BGR24, RGB24, GRAY, YUYV, UYVY, NV12, NV21, YU12 and YV12) for H264 and HEVC codecs.
H264_PROFILE	H264 encoding profile in case RAW input video frames (pixel formats BGR24, RGB24, GRAY, YUYV, UYVY, NV12, NV21, YU12 and YV12).
JPEG_QUALITY	JPEG encoding quality in case RAW input video frames (pixel formats BGR24, RGB24, GRAY, YUYV, UYVY, NV12, NV21, YU12 and YV12). Not supported by **RtspServer** class.
CODEC	Codec type for encoding RAW frames. Supported **"H264"** and **"HEVC"**.
FIT_MODE	Scaling mode. Values: **0** - fit, **1** - crop. in case RAW input video frames (pixel formats BGR24, RGB24, GRAY, YUYV, UYVY, NV12, NV21, YU12 and YV12).
OVERLAY_MODE	Overlay enable/disable in case RAW input video frames (pixel formats BGR24, RGB24, GRAY, YUYV, UYVY, NV12, NV21, YU12 and YV12). Values : **0** - disable, **1** - enable.
CYCLE_TIME_MKSEC	**Read only.** Cycle timeout, microseconds.
TYPE	Type of streamer. (**0** - unicast, **1** - multicast).
CUSTOM1	Custom parameter. Not supported by **RtspServer** class.
CUSTOM2	Custom parameter. Not supported by **RtspServer** class.
CUSTOM3	Custom parameter. Not supported by **RtspServer** class.

VStreamerParams class description

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.
    unsigned 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 RTSP server.
    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, multicastIp,
                  multicastPort, 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);
};
```

Table 6 - VStreamerParams fields description.

Parameter	Description
enable	Enable/disable RTSP server.
width	Frame width. In case processing RAW frames (pixel formats BGR24, RGB24, GRAY, YUYV, UYVY, NV12, NV21, YU12 and YV12) the library will resize input image according to this parameter value.
height	Frame height. In case processing RAW frames (pixel formats BGR24, RGB24, GRAY, YUYV, UYVY, NV12, NV21, YU12 and YV12) the library will resize input image according to this parameter value.
ip	RTSP server's ip. If the user doesn't know host IP he can set IP to **"0.0.0.0"**. It will provide default IP initialization.
port	RTSP server's port from 0 to 65535.
multicastIp	RTSP server's multicast ip. The library supports multicast IP streaming (UDP multicast). The library doesn't support requests to set particular multicast IP and port from client. If client requests particular multicast IP and port the user should set the same values in RTSP server parameters in advance.
multicastPort	RTSP server's multicast UDP port. The library doesn't support requests to set particular multicast IP and port from client. If client requests particular multicast IP and port the user should set the same values in RTSP server parameters in advance.
user	User name for auth of RTSP stream. If set to **""** the RTSP will work without auth.

Parameter	Description
password	Password name for auth of RTSP stream. If set to **""** the RTSP will work without auth.
suffix	Stream name for RTSP stream. For example "live".
minBitrateKbps	Minimum bitrate for variable bitrate encoding in case RAW input video frames. Not supported by **RtspServer** class.
maxBitrateKbps	Maximum bitrate for variable bitrate encoding in case RAW input video frames. Not supported by **RtspServer** class.
bitrateKbps	Constant bitrate encoding in case RAW input video frames (pixel formats BGR24, RGB24, GRAY, YUYV, UYVY, NV12, NV21, YU12 and YV12). The library will configure video codec paraneters if it provided by use (if user set pointer to VCodec class object).
bitrateMode	Enable/disable variable bitrate. Not supported by **RtspServer** class.
fps	Streamer's fps and also encoding fps in case RAW input video frames. Regardless of the input video frame rate, the streamer provides the specified FPS. If the FPS value is 0, the streamer provides FPS equal to the input video frame rate.
gop	Codec gop size in case in case RAW input video frames (pixel formats BGR24, RGB24, GRAY, YUYV, UYVY, NV12, NV21, YU12 and YV12) for H264 and HEVC codecs.
h264Profile	H264 encoding profile in case RAW input video frames (pixel formats BGR24, RGB24, GRAY, YUYV, UYVY, NV12, NV21, YU12 and YV12).
jpegQuality	JPEG encoding quality in case RAW input video frames (pixel formats BGR24, RGB24, GRAY, YUYV, UYVY, NV12, NV21, YU12 and YV12). Not supported by **RtspServer** class.
codec	Codec type for encoding RAW frames. Supported **"H264"** and **"HEVC"**.
fitMode	Scaling mode. Values: **0** - fit, **1** - crop. in case RAW input video frames (pixel formats BGR24, RGB24, GRAY, YUYV, UYVY, NV12, NV21, YU12 and YV12).
overlayMode	Overlay enable/disable in case RAW input video frames (pixel formats BGR24, RGB24, GRAY, YUYV, UYVY, NV12, NV21, YU12 and YV12).
cycleTimeMksec	**Read only.** Cycle (frame processing) period time, microseconds.
type	Type of streamer. (**0** - unicast, **1** - multicast).
custom1	Custom parameter. Not supported by **RtspServer** class.
custom2	Custom parameter. Not supported by **RtspServer** class.
custom3	Custom parameter. Not supported by **RtspServer** class.

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
    }
}
```

Example

First application shows how to open video file (compressed video) with **VSourceFile** library, capture video and put to RTSP server.

```
#include <iostream>
#include "RtspServer.h"
#include "VSourceFile.h"

int main(void)
{
    std::cout << "RtspServer v" <<
    cr::rtsp::RtspServer::getVersion() << std::endl;

    // Create and init RTSP server.
    cr::video::VStreamerParams params;
    params.port = 7031;
    params.ip = "127.0.0.1";
    params.suffix = "live";
    params.user = ""; // No user name.
    params.password = ""; // No password.
    params.fps = 30;
    cr::rtsp::RtspServer server;
    if(!server.initVStreamer(params))
        return -1;

    std::cout << "RTSP init string: rtsp://" << params.ip <<
    ":" << params.port << "/" << params.suffix << std::endl;

    // Init file video source.
    cr::video::VSourceFile fileSource;
    std::string initString = "out.h264;1920;1080;30";
    if(!fileSource.openVSource(initString))
        return -1;
```

```
    // Main thread.
    cr::video::Frame sourceFrame;
    while(1)
    {
        // Get new frame from video file.
        if(!fileSource.getFrame(sourceFrame))
        {
            std::cout << "Could not get frame" << std::endl;
            continue;
        }

        // Send frame to RTSP server.
        if(!server.sendFrame(sourceFrame))
            std::cout << "Could not send frame" << std::endl;
    }
    return 1;
}
```

Build and connect to your project

Typical commands to build **RtspServer** library:

```
[Linux] sudo apt-get install libopencv-dev
cd RtspServer
mkdir build
cd build
cmake ..
make
```

To install OpenCV on Windows follow oficial instructions from OpenCV website. If you want connect RtspServer 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 and copy RtspServer repository folder there. New structure of your repository:

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

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(${PARENT}_SUBMODULE_RTSP_SERVER             ON  CACHE BOOL "" FORCE)
if (${PARENT}_SUBMODULE_RTSP_SERVER)
    SET(${PARENT}_RTSP_SERVER                   ON  CACHE BOOL "" FORCE)
    SET(${PARENT}_RTSP_SERVER_EXAMPLES          OFF CACHE BOOL "" FORCE)
endif()

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

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

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

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

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

Done!