RtspServerLive555 C++ library

v2.0.1

Table of contents

Overview

The **RtspServerLive555** is a C++ library version **2.0.1** provides RTSP server. The library based on popular opensource library **Live555** (source code not included, just linked). The **RtspServerLive555** library provides simple interface to include in any C++ project. It allows you to create an RTSP server (multicast by default) without the complexity of the **Live555** library. User can set general RTSP server parameters: RTSP server port, IP, stream name, user name and password. Server works frame-by-frame: user puts video frame. **RtspServerLive555** is capable of streaming both raw frame formats (such as NV12) and compressed formats like H264. Users can provide frames that are either in a raw state or already compressed. If the frame is in a compressed format (H264, HEVC or JPEG), it bypasses any overlay and resizing processes and is sent directly to clients. However, if the frame is raw, 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 into a compressed format 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 VCodec (defaines video codec interface, source code included), VOverlay (defaines video overlay interface, source code included), **FormatConverterOpenCv** (provides methods to convert pixel formats, source code included), ConfigReader (provides methods to work with JSON files) and **OpenCV** (version 4.5 and higher). It has been specifically tested and verified for functionality on the **Linux** operating system. 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 which supports H264 and JPEG codecs.
1.0.1	22.11.2023	- Documentation updated.
1.1.0	04.12.2023	- Getting number of clients feature added. - Tracing package lost ratio feature added.
2.0.0	13.12.2023	- Interface is changed. - Streaming Raw frame support added.
2.0.1	04.01.2024	- Source code typos and styling issues fixed. - Example applications updated. - 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 3rdparty libraries
    CMakeLists.txt ------------------------ CMake file for 3rdpary folder
    ConfigReader -------------------------- folder with ConfigReader library source code
    FormatConverterOpenCv ----------------- folder with FormatConverterOpenCv library
source code
    Frame --------------------------------- folder Frame library
    VCodec -------------------------------- files of VCodec interface library
    VOverlay------------------------------- files of VOverlay interface library
src ----------------------------------------- folder with library source code
    BaseServerMediaSubsession.cpp -------- library side class source file
    BaseServerMediaSubsession.h ---------- library side class header file
    CMakeLists.txt ------------------------ CMake file
    H264Source.cpp ------------------------ library side class source file
    H264Source.h -------------------------- library side class header file
    H265Source.cpp ------------------------ library side class source file
    H265Source.h -------------------------- library side class header file
    H26xSource.cpp ------------------------ library side class source file
    H26xSource.h -------------------------- library side class header file
    MJPEGVideoSource.cpp ------------------ library side class source file
    MJPEGVideoSource.h -------------------- library side class header file
    RtspServerLive555.cpp ----------------- library main class source file
    RtspServerLive555.h ------------------- library main class header file
    RtspServerLive555Version.h ------------ header file with library version
    RtspServerLive555Version.h.in -------- service CMake file to generate version file
    Source.cpp ---------------------------- library main class source file
    Source.h ------------------------------ library main class header file
    UnicastServerMediaSubsession.cpp ----- library main class source file
    UnicastServerMediaSubsession.h ------- library main class header file
test ---------------------------------------- folder of test application
    3rdparty ------------------------------ folder with 3rdparty libraries for test
application
        VSourceFile ----------------------- folder with VSourceFile library source code.
    CMakeLists.txt ------------------------ CMake file of test application
    main.cpp ------------------------------ source C++ file of test application
examples ---------------------------------- folder for examples
    CompressedFrameStreaming -------------- folder with example for compressed video
frames
        CMakeLists.txt -------------------- CMake file of example
        main.cpp ------------------------- source code of exampole
    RawFrameStreaming --------------------- folder with example for RAW video frames
        CMakeLists.txt -------------------- CMake file of example
        main.cpp ------------------------- source code of exampole
```

Test application additionally depends on **VSourceFile** library (source code included) which provides method to read H264, HEVC files.

RtspServerLive555 class description

Class declaration

RtspServerLive555 class declared in **RtspServerLive555.h** file. Class declaration:

```
class RtspServerLive555
{
public:

    /// Class constructor.
    RtspServerLive555();

    /// Class destructor.
    ~RtspServerLive555();

    /// Get class version string.
    static std::string getVersion();

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

    /// Set param.
    bool setParam(RtspServerParam id, float value);

    /// Set param.
    bool setParam(RtspServerParam id, std::string value);

    /// Get RTSP server params.
    void getParams(RtspServerParams& params);

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

    /// Send frame.
    bool sendFrame(cr::video::Frame& frame);

    /// Close RTSP server.
    void close();
};
```

getVersion method

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

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

Method can be used without **RtspServerLive555** class instance:

```
cout << "RtspServerLive555 version: " << RtspServerLive555::getVersion() << endl;
```

Console output:

```
RtspServerLive555 version: 2.0.1
```

init method

init(...) method designed to initialize RTSP server. If server already initialized the method will reinitialize server. If RTSP server parameters changed user does not need to call init(...) method to reinitialize server. Method declaration:

```
bool init(RtspServerParams& params,
          cr::video::VCodec* codec = nullptr,
          cr::video::VOverlay* overlay = nullptr);
```

Parameter	Value
params	RtspServerParams object which includes required init parameters.
codec	VCodec object pointer in case raw frame streaming.
overlay	VOverlay object pointer .

Important note: The VCodec implementation must be designed to accept input frames in the NV12 format. The RtspServerLive555 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 correctly. 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.

setParam (string parameter) method

setParam(...) method designed to set parameter with string value. Method declaration:

```
bool setParam(RtspServerParam id, std::string value);
```

Parameter	Value
id	Parameter ID from RtspServerParam enum class.
value	Parameter value in string.

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

setParam (float parameter) method

setParam(...) method designed to set parameter with float value. Method declaration:

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

Parameter	Value
id	Parameter ID from RtspServerParam enum class.
value	Parameter value in string.

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

getParams method

getParams(...) method designed to get current parameters that are defined in form of RtspServerParams. Method declaration:

```
void getParams(RtspServerParams& params);
```

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

executeCommand method

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

```
bool executeCommand(RtspServerCommand id);
```

Parameter	Value
id	Command id from RtspServerCommand.

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

sendFrame method

sendFrame(...) method designed to send Frame to RTSP server. Method declaration:

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

Parameter	Value
frame	Frame object to send RTSP stream. H264, JPEG and raw frame formats are supported. In case of raw frame, VCodec object had to be provided to init 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 init method.

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

close method

close() method closes RTSP server if it is open. Method declaration:

```
void close();
```

Data structures

RtspServerLive555.h file defines ID's for parameters (RtspServerParam) and commands (RtspServerCommand).

RtspServerParam enum

Enum declaration:

```
enum class RtspServerParam
{
    /// Mode: 0 - disabled, 1 - enabled.
    MODE = 1,
    /// Video stream width from 8 to 8192.
    WIDTH,
    /// Video stream height from 8 to 8192.
    HEIGHT,
    /// RTSP server IP.
    IP,
    /// RTSP server port.
    PORT,
    /// RTSP server user: "" - no user.
    USER,
    /// RTSP server password: "" - no password.
```

```
    PASSWORD,
    /// RTSP stream suffix (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
};
```

Table 2 - RtspServerParam enum class description.

Parameter	Description
MODE	Enable/disable rtsp stream (0 - disabled, 1 - enabled).
WIDTH	Frame width.
HEIGHT	Frame height.
IP	Rtsp stream's ip.
PORT	Rtsp stream's port.
USER	Rtsp stream's user name for auth.
PASSWORD	Rtsp stream's password for auth.
SUFFIX	Rtsp stream's suffix.
MIN_BITRATE_KBPS	Minimum bitrate for VBR encoding in case raw frame stream.
MAX_BITRATE_KBPS	Maximum bitrate for VBR encoding in case raw frame stream.
BITRATE_KBPS	Bitrate for CBR encoding in case raw frame stream.
BITRATE_MODE	Enable/disable VBR (0 - constant bitrate, 1 - variable bitrate).
FPS	Rtsp stream's fps and also encoding fps in case raw frame stream.
GOP	Codec gop size in case of raw frame stream.

Parameter	Description
H264_PROFILE	H264 encoding profile in case of raw frame stream.
JPEG_QUALITY	JPEG encoding quality in case of raw frame stream.
CODEC	Codec type for encoding raw frames.
FIT_MODE	Scaling mode (0 - fit, 1 - cut).

RtspServerCommand enum

Enum declaration:

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

Table 3 - RtspServerCommand enum class description.

Parameter	Description
RESTART	Restarts server with last RtspServerParams
ON	Enables server if it is disabled.
OFF	Disables server if it is enabled.

RtspServerParams class description

RtspServerParams class declaration

RtspServerParams class used for video source initialization (**init** method) or to get all actual params (**getParams** method). Also **RtspServerParams** 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 RtspServerParams
{
public:
    /// RTSP server 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};
    /// RTSP server IP.
    std::string ip{"127.0.0.1"};
    /// RTSP server port.
    int port{8554};
    /// RTSP server user: "" - no user.
    std::string user{""};
    /// RTSP server password: "" - no password.
    std::string password{""};
    /// RTSP stream suffix (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};

    JSON_READABLE(RtspServerParams, enable, width, height, ip, port, user, password,
                  suffix, minBitrateKbps, maxBitrateKbps, bitrateKbps,
                  bitrateMode, fps, gop, h264Profile, jpegQuality, codec,
                  fitMode)

    /// operator =
    RtspServerParams& operator= (const RtspServerParams& src);

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

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

Table 4 - VSourceParams class fields description.

Field	Acces type	Description
enable	read/write	RTSP server mode: false - Off, true - On.
width	read/write	Video stream width from 8 to 8192.
height	read/write	Video stream height from 8 to 8192.
ip	read/write	RTSP server IP.
port	read/write	RTSP server port.
user	read/write	RTSP server user: "" - no user.
password	read/write	RTSP server password: "" - no password.
suffix	read/write	RTSP stream suffix (stream name).
minBitrateKbps	read/write	Minimum bitrate for variable bitrate mode, kbps.
maxBitrateKbps	read/write	Maximum bitrate for variable bitrate mode, kbps.
bitrateKbps	read/write	Current bitrate, kbps.
bitrateMode	read/write	Bitrate mode: 0 - constant bitrate, 1 - variable bitrate.
fps	read/write	FPS.
gop	read/write	GOP size for H264 and H265 codecs.
h264Profile	read/write	H264 profile: 0 - baseline, 1 - main, 2 - high.
jpegQuality	read/write	JPEG quality from 1 to 100% for JPEG codec.
codec	read/write	Codec type: "H264", "HEVC" or "JPEG".
fitMode	read/write	Scaling mode: 0 - fit, 1 - cut.
cycleTimeMksec	read	Cycle time, mksec. Calculated by RTSP server.

Serialization of rtsp server params

RtspServerParams class provides method **encode(...)** to serialize rtsp server params (fields of RtspServerParams class). Serialization of rtsp server params necessary in case when you need to rtsp server params via communication channels. 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, RtspServerParamsMask* mask =
nullptr);
```

Parameter	Value
data	Pointer to data buffer. Buffer size should be at least **62** bytes.

Parameter	Value
size	Size of encoded data.
bufferSize	Data buffer size.
mask	Parameters mask - pointer to **RtspServerParamsMask** structure. **RtspServerParamsMask** (declared in RtspServerLive555.h file) determines flags for each field (parameter) declared in **RtspServerParams** 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 RtspServerParamsMask structure.

RtspServerParamsMask structure declaration:

```
struct RtspServerParamsMask
{
    bool enable{true};
    bool width{true};
    bool height{true};
    bool ip{true};
    bool port{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};
};
```

Example without parameters mask:

```
// Prepare random params.
RtspServerParams in;
in.ip = "127.0.0.1";
in.port = 554;

// 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.
RtspServerParams in;
in.ip = "127.0.0.1";
in.port = 554;

// Prepare params mask.
RtspServerParamsMask mask;
mask.fitMode = 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;
```

Deserialization of rtsp server params

RtspServerParams class provides method **decode(...)** to deserialize rtsp server params (fields of RtspServerParams class). Deserialization of rtsp server params necessary in case when you need to receive rtsp server 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. 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.
RtspServerParams in;
uint8_t data[1024];
int size = 0;
in.encode(data, 1024, size);

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

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

Read params from JSON file and write to JSON file

RtspServerLive555 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.
RtspServerParams in;
cr::utils::ConfigReader inConfig;
inConfig.set(in, "RtspServerParams");
inConfig.writeToFile("TestRtspServerParams.json");

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

TestRtspServerParams.json will look like:

```
{
    "RtspServerParams": {
        "enable": 0,
        "width": 1280,
        "height": 720,
        "ip": "127.0.0.1",
        "port": 8554,
        "user": "cr",
        "password": "cr",
        "suffix": "live",
        "minBitrateKbps": 500,
        "maxBitrateKbps": 3000,
        "bitrateKbps": 2000,
        "bitrateMode": 0,
        "fps": 30,
        "gop": 30,
        "h264Profile": 0,
        "jpegQuality": 80,
        "codec": "H264",
        "fitMode": 0,
    }
}
```

Examples

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

```
#include <iostream>
#include "RtspServerLive555.h"
```

```cpp
#include "VSourceFile.h"

// Link namespaces.
using namespace std;
using namespace cr::rtsp;
using namespace cr::video;

// Entry point.
int main(void)
{
    cout << "RtspServerLive555 v" << RtspServerLive555::getVersion() << endl;
    cout << endl;

    // Init RTSP server params for encoded frames by default params.
    RtspServerParams params;
    params.port = 7031;
    params.ip = "0.0.0.0";
    params.suffix = "live";
    params.user = ""; // No user name.
    params.password = ""; // No password.
    params.fps = 30;

    // RTSP server.
    RtspServerLive555 server;
    if(!server.init(params))
    {
        cout << "Can't init RTSP server" << endl;
        return -1;
    }
    cout << "RTSP init string: rtsp://127.0.0.1:7031/live" << endl;

    // Set video source init string.
    string initString = "out.264;1280;720;30";

    // Init file video source.
    VSourceFile fileSource;
    if(!fileSource.openVSource(initString))
    {
        cout << "Can't open source file" << endl;
        return -1;
    }

    // Main thread.
    Frame sourceFrame;

    while(1)
    {
        // Get new frame from video file.
        if(!fileSource.getFrame(sourceFrame, 1000))
        {
            cout << "Frame could not get" << endl;
            continue;
        }

        // Send frame to RTSP server.
```

```
        if(!server.sendFrame(sourceFrame))
        {
            cout << "Can not send frame to rtsp server" << endl;
        }

    }

    return 1;
}
```

Second application shows how to work with RAW viudeo frames. This application includes custom implementation of video code and video overlay.

```
#include <iostream>
#include <opencv2/opencv.hpp>
#include "RtspServerLive555.h"
#include <FormatConverterOpenCv.h>



// Link namespaces.
using namespace cv;
using namespace std;
using namespace cr::rtsp;
using namespace cr::video;
using namespace std::chrono;



/// Custom video codec class based on OpenCV for JPEG encoding.
class VCodecOpenCv : public VCodec
{
public:

    /// Encode video frame.
    bool transcode(Frame& src, Frame& dst)
    {
        // Check input data.
        if (src.size == 0 || src.width == 0 || src.height == 0)
            return false;

        // Check output frame initialization.
        if (dst.width != src.width || dst.height != src.height)
            dst = Frame(src.width, src.height, Fourcc::JPEG);

        // VCodec implementations work with NV12 input
        // So RtspServerLive555 also follows same logic.
        // This means any custom VCodec implementation should accept NV12 input.
        Frame bgrFrame;
        bgrFrame.fourcc = Fourcc::BGR24;
        FormatConverterOpenCv converter;
        converter.convert(src, bgrFrame);

        // Create cv::Mat from the BGR data
        Mat bgrFrameCv(bgrFrame.height, bgrFrame.width, CV_8UC3, bgrFrame.data);
```

```
        // Set JPEG quality. Here, 95 is an example quality level.
        vector<int> compression_params;
        compression_params.push_back(IMWRITE_JPEG_QUALITY);
        compression_params.push_back(70); // Quality level here

        // Encode the image to JPG format
        vector<uchar> encodedImage;
        imencode(".jpg", bgrFrameCv, encodedImage, compression_params);

        // Copy the data from vector to uint8_t array
        copy(encodedImage.begin(), encodedImage.end(), dst.data);
        dst.size = encodedImage.size();

        return true;
    }

    // Not used
    bool setParam(VCodecParam id, float value) { return false; }

    // Not used.
    float getParam(VCodecParam id) { return -1.0; }

    // Not used.
    bool executeCommand(VCodecCommand id) { return false; }
};



/// Custom video overlay implementation. Works with YUV pixel formats.
class VOverlayCustom : public VOverlay
{
public:

    /// Overlay the information on the video.
    bool overlay(Frame& frame, void* data = nullptr)
    {
        // Put the text on the frame.
        static int counter = 0;
        Mat yuvImg(frame.height, frame.width, CV_8UC3, frame.data);
        putText(yuvImg, to_string(counter++), Point(60, 60),
                FONT_HERSHEY_SIMPLEX, 2.5, Scalar(76, 84, 255), 2);
        return true;
    }
};



// Entry point.
int main(void)
{
    cout << "RtspServerLive555 v" << RtspServerLive555::getVersion() << endl;
    cout << endl;

    // Set initial RTSP server params.
```

```
    RtspServerParams params;
    params.port = 7031;
    params.ip = "0.0.0.0"; // For any IP.
    params.suffix = "live";
    params.user = "";       // No user name.
    params.password = "";   // No password.
    params.fps = 30;
    params.codec = "JPEG";
    params.width = 1280;
    params.height = 720;
    params.fitMode = 0;

    // Create and init RTSP server.
    RtspServerLive555 server;
    if(!server.init(params, new VCodecOpenCv(), new VOverlayCustom()))
    {
        cout << "Can't init RTSP server" << endl;
        return -1;
    }
    cout << "RTSP init string: rtsp://127.0.0.1:7031/live" << endl;

    // Main loop.
    int sourceFrameWidth = 1280;
    int sourceFrameHeight = 1024;
    Mat sourceFrame(sourceFrameHeight, sourceFrameWidth, CV_8UC3);
    int frameId = 0;
    while(1)
    {
        // Create new artificial frame.
        memset(sourceFrame.data, 60, sourceFrameWidth * sourceFrameHeight * 3);
        putText(sourceFrame, to_string(frameId++), Point(300, 300),
                FONT_HERSHEY_SIMPLEX, 6, Scalar(255, 255, 0), 5);

        // Prepare Frame object from OpenCV.
        Frame bgrFrame;
        bgrFrame.width = sourceFrame.size().width;
        bgrFrame.height = sourceFrame.size().height;
        bgrFrame.size = bgrFrame.width * bgrFrame.height * 3;
        bgrFrame.data = sourceFrame.data;
        bgrFrame.fourcc = Fourcc::BGR24;

        // Send  RAW frame to RTSP server.
        if(!server.sendFrame(bgrFrame))
        {
            cout << "Can not send frame to rtsp server" << endl;
            continue;
        }

        // Wait for aprox 30 fps.
        this_thread::sleep_for(milliseconds(33));
    }

    return 1;
}
```

Dependencies

The **RtspServerLive555** is a C++ library uses the **Live555** library, which in turn relies on OpenSSL for secure network communication. To effectively utilize the **RtspServerLive555** library, it is essential to ensure that both Live555 and OpenSSL are correctly installed and configured on your system. Also library uses **OpenCV** library (version 4.5 and higher).

How to install OpenSSL

- Run command :

```
sudo apt-get install libssl-dev
```

How to install Live555 by using package manager

- Run command :

```
sudo apt-get install liblivemedia-dev
```

If you get error from package manager, go to next step and install live555 from source code.

Important note : There will be a compilation error if your system has live555 from source code installation and package manager installation at the same time. In order to solve problem you should remove one of them. For example:

```
sudo apt-get remove liblivemedia-dev
```

How to install Live555 from source code

- First you need to download Live555 from **official website**.
- Extract .tar.gz file.

```
tar -xz live.(version).tar.gz
```

- Run **genMakefiles** command with proper OS name. You check supported OS in live folder see files config.OSname .

```
cd live
```

```
./genMakefiles <os-platform>
```

for example:

```
./genMakefiles linux-64bit
```

if you are getting compile error. Add -std=c++20 -DNO_STD_LIB flags to CPLUSPLUS_FLAGS param in **config.os-platform** for example **config.linux-64-bit**:

```
COMPILE_OPTS =      $(INCLUDES) -fPIC -I/usr/local/include -I. -O2 -
DSOCKLEN_T=socklen_t -D_LARGEFILE_SOURCE=1 -D_FILE_OFFSET_BITS=64
C =         c
C_COMPILER =        cc
C_FLAGS =       $(COMPILE_OPTS)
CPP =           cpp
CPLUSPLUS_COMPILER =    c++
CPLUSPLUS_FLAGS =   $(COMPILE_OPTS) -Wall -DBSD=1 -Wno-deprecated -std=c++20 -
DNO_STD_LIB
OBJ =           o
LINK =          c++ -o
LINK_OPTS =     -L.
CONSOLE_LINK_OPTS = $(LINK_OPTS)
LIBRARY_LINK =      ar cr
LIBRARY_LINK_OPTS =
LIB_SUFFIX =            a
LIBS_FOR_CONSOLE_APPLICATION = -lssl -lcrypto
LIBS_FOR_GUI_APPLICATION =
EXE =
```

- Run make command

```
make
```

- Install to system

```
sudo make install
```

How to install opencv

- Run command :

```
sudo apt-get install libopencv-dev
```

Build and connect to your project

Typical commands to build **RtspServerLive555** library:

```
cd RtspServerLive555
mkdir build
cd build
cmake ..
make
```

If you want connect RtspServerLive555 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 **RtspServerLive555** as submodule by command (if you don't have access to online repository just compy repository folder):

```
cd <your respository folder>
git submodule add https://github.com/ConstantRobotics-Ltd/RtspServerLive555.git
3rdparty/RtspServerLive555
```

In you repository folder will be created folder **3rdparty/RtspServerLive555** which contains files of **RtspServerLive555** repository. New structure of your repository:

```
CMakeLists.txt
src
    CMakeList.txt
    yourLib.h
    yourLib.cpp
3rdparty
    RtspServerLive555
```

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(${PARENT}_SUBMODULE_RTSP_SERVER_LIVE555             ON  CACHE BOOL "" FORCE)
if (${PARENT}_SUBMODULE_RTSP_SERVER_LIVE555)
    SET(${PARENT}_RTSP_SERVER_LIVE555                   ON  CACHE BOOL "" FORCE)
```

```
    SET(${PARENT}_RTSP_SERVER_LIVE555_TEST              OFF CACHE BOOL "" FORCE)
    SET(${PARENT}_RTSP_SERVER_LIVE555_EXAMPLES          OFF CACHE BOOL "" FORCE)
endif()

################################################################################
## INCLUDING SUBDIRECTORIES
## Adding subdirectories according to the 3rd-party configuration
################################################################################
if (${PARENT}_SUBMODULE_RTSP_SERVER_LIVE555)
    add_subdirectory(RtspServerLive555)
endif()
```

File **3rdparty/CMakeLists.txt** adds folder **RtspServerLive555** 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
    RtspServerLive555
```

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 need to add following part in main **CMakeLists.txt** file of your repository in order to include Live555 library properly. This part should be located before ***add_subdirectory(3rdparty)*** in main **CMakeLists.txt**.

```
if(EXISTS /usr/include/liveMedia)
    include_directories(/usr/include/BasicUsageEnvironment)
    include_directories(/usr/include/groupsock)
    include_directories(/usr/include/UsageEnvironment)
    include_directories(/usr/include/liveMedia)
    set(live555PackageInstallation 1)
    add_definitions(-Dlive555PackageInstallation=${live555PackageInstallation})
else()
    include_directories(/usr/local/include/BasicUsageEnvironment)
    include_directories(/usr/local/include/groupsock)
    include_directories(/usr/local/include/UsageEnvironment)
    include_directories(/usr/local/include/liveMedia)
endif()
```

Next you have to include RtspServerLive555 library in your **src/CMakeLists.txt** file:

```
target_link_libraries(${PROJECT_NAME} RtspServerLive555)
```

Done!