VOutputV4L2 C++ library

v1.0.1

Table of contents

Overview

VOutputV4L2 C++ library provide interface to write video frame to V4L2 video devices in Linux OS. An example of a video device is v4l2loopback device (virtual video devices). Normal (v4l2) applications will read these devices as if they were ordinary video devices, but the video will not be read from e.g. a capture card but instead it is generated by user's application. Using the library user's application can be as video source for third-party processes. The library provides simple interface. It depends on open source Frame library (describes video frame structure and pixel formats) and open source Logger library (provides method to write logs). The library supports C++17 standard and works only on Linux with V4L2 API support.

Versions

Table 1 - Library versions.

Version	Release date	What's new
1.0.0	10.08.2023	First version.

Version	Release date	What's new
1.0.1	09.01.2024	- Examples updated. - Code optimized. - Documentation updated.

Library files

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

```
CMakeLists.txt ---------------- Main CMake file.
3rdparty ---------------------- Folder with third-party libraries.
    CMakeLists.txt ------------ CMake file to include third-party libraries.
    Logger -------------------- Folder with files of Logger library.
    Frame --------------------- Folder with files of Frame library.
src --------------------------- Folder with library source code.
    CMakeLists.txt ------------ CMake file.
    VOutputV4L2.h ------------- Main library header file.
    VOutputV4L2version.h ------ Header file with library version.
    VOutputV4L2version.h.in --- File for CMake to generate version header.
    VOutputV4L2.cpp ----------- C++ implementation file.
test -------------------------- Folder for test application files.
    CMakeLists.txt ------------ CMake file for test application.
    main.cpp ------------------ Source C++ file of test application.
example ----------------------- Folder for example application files.
    CMakeLists.txt ------------ CMake file for example application.
    main.cpp ------------------ Source C++ file of example application.
```

Additionally test application depends on OpenCV library to provide user interface.

VOutputV4L2 class description

VOutputV4L2 class declaration

VOutputV4L2 class declared in **VOutputV4L2.h** file. Class declaration:

```
class VOutputV4L2
{
public:

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

    /// Class constructor.
    VOutputV4L2();
```

```
    /// Class destructor.
    ~VOutputV4L2();

    /// Open device.
    bool open(std::string device);

    /// Write video frame to device.
    bool write(Frame& frame);

    /// Set log level.
    void setLogLevel(cr::utils::PrintFlag flag);

    /// Close device.
    void close(void);
};
```

getVersion method

getVersion() method returns string of current version of **VOutputV4L2** class. Method declaration:

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

Method can be used without **VOutputV4L2** class instance:

```
cout << "VOutputV4L2 class version: " << VOutputV4L2::getVersion() << endl;
```

Console output:

```
VOutputV4L2 class version: 1.0.1
```

open method

open(...) method initializes video output device, checks capabilities of device and turns video stream on.

```
bool open(string device);
```

Parameter	Value
device	full V4L2 video device name. For example: "/dev/video4".

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

write method

write(...) method writes video frame video device.

```
bool write(Frame& frame);
```

Parameter	Value
frame	Frame class object. Supported only formats (Fourcc enum of Frame class): RGB24, BGR24, YUYV, UYVY, GRAY, YUV24, NV12, NV21, YU12, YV12, HEVC, H264 and JPEG. Some devices may not support particular formats.

Returns: TRUE if the video frame is written to device or FALSE if not.

setLogLevel method

setLogLevel(...) method designed to set new video source parameters value. Method declaration:

```
void setLogLevel(cr::utils::PrintFlag flag);
```

Parameter	Description
flag	Logger flag which defined in **Logger.h** from Logger library: DISABLE - disabled. CONSOLE - print to console (terminal) only. FILE - print to file only if save params was set. CONSOLE_AND_FILE - print to file and console.

close method

close() method closes output device and turn stream off. Method declaration:

```
void close();
```

Build and connect to your project

Typical commands to build **VOutputV4L2** library:

```
cd VOutputV4L2
git submodule update --init --recursive
mkdir build
cd build
cmake ..
make
```

If you want connect **VOutputV4L2** library to your CMake project as source code you can make follow. For example, if your repository has structure:

```
CMakeLists.txt
src
    CMakeList.txt
    yourLib.h
    yourLib.cpp
```

Create folder **3rdparty** in your repository. Copy repository folder **VOutputV4L2** to **3rdparty** folder. New structure of your repository:

```
CMakeLists.txt
src
    CMakeList.txt
    yourLib.h
    yourLib.cpp
3rdparty
    VSourceV4L2
```

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(${PARENT}_SUBMODULE_VOUTPUT_V4L2                    ON  CACHE BOOL "" FORCE)
if (${PARENT}_SUBMODULE_VOUTPUT_V4L2)
    SET(${PARENT}_VOUTPUT_V4L2                          ON  CACHE BOOL "" FORCE)
    SET(${PARENT}_VOUTPUT_V4L2_TEST                     OFF CACHE BOOL "" FORCE)
    SET(${PARENT}_VOUTPUT_V4L2_EXAMPLE                  OFF CACHE BOOL "" FORCE)
endif()

###############################################################################
## INCLUDING SUBDIRECTORIES
## Adding subdirectories according to the 3rd-party configuration
###############################################################################
```

```
if (${PARENT}_SUBMODULE_VOUTPUT_V4L2)
    add_subdirectory(VOutputV4L2)
endif()
```

File **3rdparty/CMakeLists.txt** adds folder **VOutputV4L2** to your project and excludes test application (VOutputV4L2 class test applications) from compiling. Your repository new structure will be:

```
CMakeLists.txt
src
    CMakeList.txt
    yourLib.h
    yourLib.cpp
3rdparty
    CMakeLists.txt
    VOutputV4L2
```

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 VOutputV4L2 library in your **src/CMakeLists.txt** file:

```
target_link_libraries(${PROJECT_NAME} VOutputV4L2)
```

NOTE: You should run your application which includes VOutputV4L2 library with root (sudo) privileges.

Simple example

Below is the code for a simple application for streaming generated video frames to /dev/video4 device.

```
#include <iostream>
#include "VOutputV4L2.h"

int main(void)
{
    // Open video device.
    cr::video::VOutputV4L2 videoOutput;
    if(!videoOutput.open("/dev/video4"))
        return -1;

    // Create YUV frame with certain resolution.
    cr::video::Frame frame(640, 480, cr::video::Fourcc::YUV24);

    // Main loop.
    uint8_t value = 0;
    while (true)
    {
        // Fill image by color.
        memset(frame.data, value++, frame.size);
```

```
        // Write frame to device.
        if(!videoOutput.write(frame))
            std::cerr << "Cant' write frame" << std::endl;

        // 33ms delay for 30 fps stream
        std::this_thread::sleep_for(std::chrono::milliseconds(33));
    }
    return 1;
}
```