Joystick C++ library

v2.0.2

Table of contents

Overview

Joystick is a C++ library which assists developers in seamlessly interfacing with various types of joysticks, all without relying on third-party dependencies. The library compatible with Linux (uses Linux native joysticks interface) and Windows (uses WinAPI). The library doesn't have third party dependencies. The library is CMake project and uses C++17 standard.

Versions

Table 1 - Library versions.

Version	Release date	What's new
1.0.0	27.11.2022	First version

Version	Release date	What's new
2.0.0	04.10.2023	- Interface changed. - Class name changed from JoystickController to Joystick. - SDL2 library excluded. No more third-party dependencies. - Documentation updated.
2.0.1	04.10.2023	- Method description updated.
2.0.2	15.10.2023	- Documentation updated. - Files structure changed. - Variables names changed.

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
src ------------------------------ folder with library source code
    CMakeLists.txt --------------- CMake file
    Joystick.h ------------------- main library header file
    JoystickVersion.h ------------ header file with library version
    JoystickVersion.h.in --------- file for CMake to generate version header
    Joystick.cpp ----------------- C++ implementation file
test ----------------------------- folder for demo application files
    CMakeLists.txt --------------- CMake file for test application
    main.cpp --------------------- source C++ file of demo application
test ----------------------------- folder for example application files
    CMakeLists.txt --------------- CMake file for example application
    main.cpp --------------------- source C++ file of example application
```

Joystick class description

Joystick class declaration

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

```
/**
 * @brief Joystick class.
 */
class Joystick
{
public:

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

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

    /// Class destructor.
    ~Joystick();

    /// Find available joysticks.
    std::vector<JoystickInfo> findJoysticks();

    /// Open joystick.
    bool openJoystick(int id);

    /// Close joystick.
    void closeJoystick();

    /// Check status of a button.
    bool getButtonState(int buttonId);

    /// Get hat value.
    int getHatValue();

    /// Get axis value.
    int getAxisValue(int axisId);
};
```

getVersion method

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

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

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

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

Console output:

```
Joystick class version: 2.0.2
```

findJoysticks method

findJoysticks() method detects connected joysticks and provides information about joysticks. When called, this method returns a vector of **JoystickInfo** structures, each containing properties of the detected joysticks. Method declaration:

```
std::vector<JoystickInfo> findJoysticks();
```

Returns: vector of **JoystickInfo** structures. **JoystickInfo** structure defined in Joystick.h file. Structure declaration:

```
/// Struct for joystick informations.
struct JoystickInfo
{
    /// Joystick name.
    std::string name;
    /// Joystick id (from 0).
    int id;
    /// Number of buttons.
    int numButtons;
    /// Number of axes.
    int numAxes;
};
```

Table 2 - Joystick properties description.

Parameter	Description
name	Name of joystick.
id	Joystick id from 0 to max number of connected joystick -1.
numButtons	Number of available buttons.
numAxes	Number of supported axes.

openJoystick method

openJoystick(...) method opens joystick which is specified by id. Method declaration:

```
bool openJoystick(int id);
```

Parameter	Value
id	joystick id from zero to max -1 connected joysticks which can be determined by length of vector returned by **findJoystick()** method.

Returns: TRUE if it success or FALSE if not.

closeJoystick method

closeJoystick() method closes an opened joystick. Method declaration:

```
void closeJoystick();
```

getButtonState method

getButtonState(...) method serves the purpose of reading state of a button from a joystick. Method declaration:

```
bool getButtonState(int buttonId);
```

Parameter	Value
buttonId	Button id which can be from zero to number of buttons in JoystickInfo struct.

Returns: TRUE if button is pressed or FALSE if not.

getHatValue method

getHatValue() method is designated to read hat value from a joystick. Method declaration:

```
int getHatValue();
```

Returns: hat value from 0 to 8. Return values:

Value	hat possition
0	Not pressed
1	Up
2	Up and right
3	Right
4	Right and down
5	Down
6	Down and left
7	Left
8	Left and up

getAxisValue method

getAxisValue() method is designated to read axis value from a joystick. Method declaration:

```
int getAxisValue(int axisId);
```

Parameter	Value
axisId	Axix id which can be from zero to number of buttons -1 in JoystickInfo struct.

Returns: Axis position. Range depends on joystick but generally between 32767 and -32767.

Build and connect to your project

Typical commands to build **Joystick** library:

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

If you want connect **Joystick** 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 **Joystick** as submodule by commands (or copy repository files):

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

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

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

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(${PARENT}_SUBMODULE_JOYSTICK                        ON  CACHE BOOL "" FORCE)
if (${PARENT}_SUBMODULE_JOYSTICK)
    SET(${PARENT}_JOYSTICK                              ON  CACHE BOOL "" FORCE)
    SET(${PARENT}_JOYSTICK_TEST                         OFF CACHE BOOL "" FORCE)
    SET(${PARENT}_JOYSTICK_EXAMPLE                      OFF CACHE BOOL "" FORCE)
endif()

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

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

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

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

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

Done!

Simple example

This example illustrates the open a joystick and read button values.

```
#include <iostream>
#include <thread>
#include <chrono>
#include "Joystick.h"

int main(void)
{
    // Create joystick controller.
    cr::utils::Joystick joyStickController;

    // Read number of connected joysticks.
    std::vector<cr::utils::JoystickInfo> joysticks;
    joysticks = joyStickController.findJoysticks();

    // Open first joystick
    joyStickController.openJoystick(joysticks.at(0).id);

    // Print joystick infos.
    std::cout << "Joystick name : " << joysticks.at(0).name << std::endl;
    std::cout << "Number of button : " << joysticks.at(0).numButtons << std::endl;
    std::cout << "Number of axes : " << joysticks.at(0).numAxes << std::endl;

    // Read all avaliable buttons one by one
    while (true)
    {
        for (int i = 0; i < joysticks.at(0).numButtons; ++i)
        {
            if (joyStickController.getButtonState(i) == true)
                std::cout << "Button : " << i << " pressed" << std::endl;
        }

        std::this_thread::sleep_for(std::chrono::milliseconds(100));
    }
    return 0;
}
```