SerialPort C++ library

v3.0.2

Table of contents

Overview

SerialPort C++ library provides simple interface to work with serial ports. **SerialPort.h** file contains declaration of **SerialPort** C++ class. **SerialPort** has functions: open, write data and read data from serial port. SerialPort library also provides applications to test communication with any equipment (send data and check response). The library requires C++17 standard. The library doesn't have any third-party dependency. The library compatible with Linux and Window OS.

Versions

Table 1 - Library versions.

Version	Release date	What's new
1.0.0	14.10.2021	First version
2.0.0	20.05.2022	- Interface changed. - Added method to RTS/CTS hardware flow control.
2.1.0	10.07.2022	- Code optimized.
2.2.0	22.03.2023	- Code optimized. - Documentation updated.
2.3.0	24.04.2023	- Added new test application.
2.4.0	19.06.2023	- Added new test application (SerialPortStringTester).
2.5.0	26.06.2023	- Updated test applications (SerialPortStringTester and SerialPortTester combined into one application).
3.0.0	27.06.2023	- Changed interface (added new methods read(...) and write(...) instead of readData(...) and sendData(...)).
3.0.1	19.07.2023	- Fixed compiling error in methods read(...) and write(...) for Linux.
3.0.2	17.12.2023	- Methods description updated. - Documentation updated. - Default branch name changed from "main" to "master".

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
    SerialPort.h ----------------- main library header file
    SerialPortVersion.h ---------- header file with library version
    SerialPortVersion.h.in ------- file for CMake to generate version header
    SerialPort.cpp --------------- C++ implementation file
examples ------------------------- folder for examples
    CMakeLists.txt --------------- CMake file to include examples
    SerialPortDataReceiver ------- folder with data receiver example
        CMakeLists.txt ----------- CMake file of data receiver example
        main.cpp ----------------- source code of data receiver example
    SerialPortDataSender --------- folder with data sender example
```

```
        CMakeLists.txt ----------- CMake file of data sender example
        main.cpp ----------------- source code of data sender example
    SerialPortTester ------------- folder with serial port tester application
        CMakeLists.txt ----------- CMake file of serial port tester application
        main.cpp ----------------- source code of serial port tester application
```

SerialPort class description

Class declaration

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

```
class SerialPort
{
public:

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

    /// Class constructor.
    SerialPort();

    /// Class destructor.
    ~SerialPort();

    /// Open serial port.
    bool open(std::string file, unsigned int baudrate,
              unsigned int timeoutMsec = 100, const char *mode = "8N1");

    /// Read data from serial port.
    int read(uint8_t *buf, uint32_t size);

    /// Write data to serial port.
    int write(uint8_t *buf, uint32_t size);

    /// Get open status.
    bool isOpen();

    /// Close serial port.
    void close();

    /// Set RTS/CTS hardware flow control.
    bool setFlowControl(bool enable);
};
```

getVersion method

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

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

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

```
cout << "Serial port version: " << SerialPort::getVersion() << endl;
```

Console output:

```
Serial port version: 3.0.2
```

open method

open(...) method opens serial port. If serial port already open the method firstly will close serial port and will try open again according to method's parameters. Method declaration:

```
bool open(std::string file, unsigned int baudrate, unsigned int timeoutMsec = 100, const
char *mode = "8N1");
```

Parameter	Value
file	Full serial port file name (serial device name). In **Windows** serial ports are named **\\.\COM**. In typical **UNIX** style, serial ports are represented by files within the operating system. These files usually pop-up in `/dev/`, and begin with the name `tty*`. Common names are: **/dev/ttyACM0**- ACM stands for the ACM modem on the USB bus. Arduino UNOs (and similar) will appear using this name. **/dev/ttyPS0** - Xilinx Zynq FPGAs running a Yocto-based Linux build will use this name for the default serial port that Getty connects to. **/dev/ttyS0** - Standard COM ports will have this name. These are less common these days with newer desktops and laptops not having actual COM ports. **/dev/ttyUSB0** *- Most USB-to-serial cables will show up using a file named like this. **/dev/pts/0** - A pseudo terminal. These can be generated with socat.
baudrate	Baudrate e.g. 2400, 4800, 9600, 19200, 38400, 57600, 115200 etc.
timeoutMsec	Timeout in milliseconds for reading data from serial port. When user call **readData(...)** method it will wait **timeout** msec and will return all data from input serial port buffer.
mode	Mode. Default "8N1" most common. Always 3 symbols: 1 - Number of bits (8, 7, 6, 5), 2 - parity (N, E, O), 3 - number of stop bits (1 or 2).

Returns: TRUE if the serial port open or FALSE if not.

read method

read(...) method reads data from serial port. Method will wait **timeoutMsec** (set by user in **open(...)** method) and will return all data (<= requested amount of data) from input serial port buffer. If you don't want to wait and just check data in serial port or if you want to use different timeouts to wait data set **timeout = 0** in open(...) method. Method declaration:

```
int read(uint8_t *buf, uint32_t size);
```

Parameter	Value
buf	Pointer to buffer.
size	Size of buffer. Size of data which you want to read from serial port. Method will return **<= size** bytes from input serial port buffer.

Returns: number of received bytes **<= size** or **0** if no data in input serial port buffer or **-1** if serial port not open.

write method

write(...) method intended to send data to serial port. Method declaration:

```
int write(uint8_t *buf, uint32_t size);
```

Parameter	Value
buf	Pointer to buffer with data to send.
size	Number of bytes to send.

Returns: number of bytes sent **<= size** or **-1** if serial port not open.

isOpen method

isOpen() method intended to obtain serial port connection status. Method declaration:

```
bool isOpen();
```

Returns: TRUE is the serial port open or FALSE if not.

close method

close() method intended to close serial port. Method declaration:

```
void close();
```

setFlowControl method

setFlowControl(...) method intended to set RTS/CTS hardware flow control. RTS / CTS Flow Control is **another flow control mechanism that is part of the RS232 standard**. It makes use of two further pins on the RS232 connector, RTS (Request to Send) and CTS (Clear to Send). These two lines allow the receiver and the transmitter to alert each other to their state. Method declaration:

```
bool setFlowControl(bool enable);
```

Parameter	Value
enable	Enable RTS/CTS hardware flow control flag: **true** - enable, **false** - disable.

Returns: TRUE if mode was changed or FALSE.

Examples

Data sender example

```
#include <iostream>
#include <chrono>
#include <ctime>
#include <thread>
#include "SerialPort.h"

/// Link namespaces.
using namespace std;
using namespace cr::clib;
using namespace std::chrono;

// Entry point.
int main(void)
{
    cout<< "================================================" << endl;
    cout<< "SerialPortDataSender " << SerialPort::getVersion() << endl;
    cout<< "================================================" << endl;
    cout<< endl;

    // Enter serial port num.
    int portNum = 0;
```

```
    cout << "Enter serial port num: ";
    cin >> portNum;

    // Enter serial port baudrate.
    int portBaudrate = 0;
    cout << "Enter serial port baudrate: ";
    cin >> portBaudrate;

    // Enter numer of bytes.
    int numBytesToSend = 0;
    cout << "Enter num bytes to send: ";
    cin >> numBytesToSend;

    // Enter sending data period ms.
    int cyclePeriodMs = 0;
    cout << "Enter sending data period ms: ";
    cin >> cyclePeriodMs;

    // Open serial port.
#if defined(linux) || defined(__linux) || defined(__linux__)|| defined(__FreeBSD__)
    std::string portName = "/dev/ttyS" + std::to_string(portNum);
#elif defined(_WIN32) || defined(__WIN32__) || defined(WIN32)
    string portName = "\\\\.\\COM" + to_string(portNum);
#endif

    // Init serial port.
    SerialPort serialPort;
    if (!serialPort.open(portName.c_str(), portBaudrate))
    {
        cout << "ERROR: Serial port not open. Exit." << endl;
        this_thread::sleep_for(seconds(1));
        return -1;
    }

    // Init variables.
    uint8_t* outputData = new uint8_t[numBytesToSend];

    // Main loop.
    chrono::time_point<system_clock> startTime = system_clock::now();
    while (true)
    {
        // Prepare random data.
        for (int i = 0; i < numBytesToSend; ++i)
            outputData[i] = (uint8_t)(rand() % 255);

        // Send data.
        std::cout << serialPort.write(outputData, numBytesToSend) <<
                     " bytes sent" << std::endl;

        // Wait according to parameters.
        int waitTime = (int)duration_cast<std::chrono::milliseconds>(
                    system_clock::now() - startTime).count();
        waitTime = cyclePeriodMs - waitTime;
        if (waitTime > 0)
            this_thread::sleep_for(milliseconds(waitTime));
```

```
        startTime = system_clock::now();
    }

    return 1;
}
```

Data receiver example

```
#include <iostream>
#include <chrono>
#include <ctime>
#include <thread>
#include "SerialPort.h"

/// Link namespaces.
using namespace std;
using namespace cr::clib;
using namespace std::chrono;

// Entry point.
int main(void)
{
    cout<< "================================================" << endl;
    cout<< "SerialPortDataReceiver " << SerialPort::getVersion()<< endl;
    cout<< "================================================" << endl;
    cout<< endl;

    // Enter serial port num.
    int portNum = 0;
    cout << "Enter serial port num: ";
    cin >> portNum;

    // Enter serial port baudrate.
    int portBaudrate = 0;
    cout << "Enter serial port baudrate: ";
    cin >> portBaudrate;

    // Enter wait data timeout.
    int waitDataTimeoutMs = 0;
    cout << "Enter wait data timeout ms: ";
    cin >> waitDataTimeoutMs;

    // Open serial port.
#if defined(linux) || defined(__linux) || defined(__linux__)|| defined(__FreeBSD__)
    std::string portName = "/dev/ttyS" + std::to_string(portNum);
#elif defined(_WIN32) || defined(__WIN32__) || defined(WIN32)
    string portName = "\\\\.\\COM" + to_string(portNum);
#endif

    // Init serial port.
    SerialPort serialPort;
    if (!serialPort.open(portName.c_str(), portBaudrate, waitDataTimeoutMs))
    {
```

```
        cout << "ERROR: Serial port not open. Exit" << endl;
        this_thread::sleep_for(seconds(1));
        return -1;
    }

    // Init variables.
    const uint16_t inputDataSize = 1024;
    uint8_t inputData[inputDataSize];

    // Main loop.
    while (true)
    {
        // Read data.
        int bytes = serialPort.read(inputData, inputDataSize);

        // Check input data size.
        if (bytes <= 0)
            cout << "No input data" << endl;
        else
            cout << bytes << " bytes read." << endl;
    }
}
```

Build and connect to your project

Typical commands to build **SerialPort** library (on Linux OS):

```
git clone https://github.com/ConstantRobotics-Ltd/SerialPort.git
cd SerialPort
mkdir build
cd build
cmake ..
make
```

If you want connect **SerialPort** 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 **SerialPort** as submodule by command (or just copy files):

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

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

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

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(${PARENT}_SUBMODULE_SERIAL_PORT                     ON  CACHE BOOL "" FORCE)
if (${PARENT}_SUBMODULE_SERIAL_PORT)
    SET(${PARENT}_SERIAL_PORT                           ON  CACHE BOOL "" FORCE)
    SET(${PARENT}_SERIAL_PORT_EXAMPLES                  OFF CACHE BOOL "" FORCE)
endif()

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

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

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

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

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

Done!

SerialPortTester application

SerialPortTester is application designed to test communication with any equipment via serial port. The app allows the user to send any data (manual input) to the serial port and check the response from the equipment. The app allows data entry in both HEX and string format (for ASCII protocols). The application supports Windows and Linux OS. How to use:

Copy executable file SerialPortTester and make it executable:

```
sudo chmod +x SerialPortTester
```

Run application from sudo (the application doesn't have params):

```
sudo ./SerialPortTester
```

You will see dialog to enter serial port name. On **Windows OS** you should set COM port num (the application will make serial port name according to Windows format **"\\\\.\\COM" + to_string(portNum)**):

```
================================================
Serial port tester v3.0.2
================================================

Set COM port num (1,2,3,...): 2
```

On **Linux OS** you have to type full serial port name: **/dev/ttyUSB0,1(N)** (for USB to serial adapters), **/dev/ttyS0,1(N)** (for build in serial ports), **/dev/serial/by-id/(port name)** (for USB to serial adapters) and push "Enter" on keyboard:

```
================================================
Serial port tester v3.0.2
================================================

Set serial port name: /dev/ttyUSB0
```

After you have to set baudrate:

```
================================================
Serial port tester v3.0.1
================================================

Set serial port name: /dev/ttyUSB0
```

After you have to set baudrate and push "Enter" on keyboard:

```
================================================
Serial port tester v3.0.1
================================================

Set serial port name: /dev/ttyUSB0
Set baudrate: 115200
```

After you have to set chose mode: string mode (you will be able print text for ASCII protocols) or HEX mode (you will be able to print HEX values):

```
================================================
Serial port tester v3.0.1
================================================

Set serial port name: /dev/ttyUSB0
Set baudrate: 115200
Chose mode (1 - string, 0 - HEX): 0
```

After you will be able to enter message to send. In HEX mode you have to print string in HEX format and push push "Enter" on keyboard. The application will send your message to serial port and will wait 2 sec. After 2 sec the application will check and will show response or will show **"ERROR: No response from serial port"** message:

```
================================================
Serial port tester v3.0.2
================================================

Set serial port name: /dev/serial/by-id/usb-FTDI_USB-RS232_Cable_FT5MJ4PE-if00-port0
Set baudrate: 115200
Chose mode (1 - string, 0 - HEX): 0
Enter HEX string (e.g. AAF00A): AA040170001FEBAA
[TX]: aa 4 1 70 0 1f eb aa
[RX]: 55 17 70 33 46 54 49 49 36 34 30 30 30 31 30 30 30 30 30 58 45 50 4e 0 91 eb aa
Enter HEX string (e.g. AAF00A): AA0401710020EBAA
[TX]: aa 4 1 71 0 20 eb aa
[RX]: 55 17 71 33 42 31 31 36 30 31 34 31 0 0 0 0 0 0 0 0 0 0 0 0 b0 eb aa
```

```
Enter HEX string (e.g. AAF00A):
```

In string mode the application in additional to input HEX data will show string of response.

```
================================================
Serial port tester v3.0.2
================================================

Set serial port name: /dev/serial/by-id/usb-FTDI_USB-RS232_Cable_FT5MJ4PE-if00-port0
Set baudrate: 115200
Chose mode (1 - string, 0 - HEX): 1
Enter string: TestMessage
[TX]: 54 65 73 74 4d 65 73 73 61 67 65 d a
ERROR: No response from serial port
Enter string: ZR
[TX]: 5a 52 d a
[RX]: 5a 52 3d 31 d a  :  ZR=1
```

In string mode the application adds symbols **'\r'** and **'\n'** at the end of a string automatically.