Serialink Library

Generated by Doxygen 1.9.1

Chapter 1

Serialink Library

Serialink is a library for serial communication designed to support and simplify software development in Linux environments. This library provides a straightforward interface and configuration, making it easy to integrate into various applications. Additionally, Serialink supports advanced serial communication, including communication using specific protocols (framed data).

1.1 Installation

Follow these steps to install the necessary packages and dependencies.

1.1.1 Installing Compiler and CMake

1. Install the Compiler:

```
sudo apt install build-essential g++ binutils
```

1. Install CMake:

```
sudo apt install make cmake
```

1.1.2 Installing Google Test

1. Clone repository:

```
git clone https://github.com/google/googletest.git -b v1.14.0
```

1. Navigate to the cloned repository directory:

```
cd googletest
```

1. Build the Google Test Library:

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

1. Install Google Test Library:

```
sudo make install
```

1. Return to the root directory:

```
cd ../../
```

1.2 Building the Library for Test

1. Clone the main repository:

```
git clone https://github.com/JayaWikrama/Serialink.git
```

1. Navigate to the project directory:

```
cd Serialink
```

1. Initialize and update all Git submodules:

```
git submodule init
git submodule update --init --recursive
```

1. If you want to use the latest source code (same as the remote repository), run:

```
git submodule update --remote
```

1. Create a build directory:

```
mkdir build
cd build
```

1. Configure CMake:

```
cmake ..
```

1. Build the library:

```
make
```

1. Run the unit tests to ensure all functions are working correctly:

```
./Serialink-test
```

Sample test output:

```
[==========] Running 35 tests from 2 test suites.
[----------] Global test environment set-up.
[----------] 26 tests from SerialinkSimpleTest
[ RUN      ] SerialinkSimpleTest.DefaultConstructor_1
Virtual Serial Name (slave): /dev/pts/1
Virtual serial port ready
[       OK ] SerialinkSimpleTest.DefaultConstructor_1 (0 ms)
[ RUN      ] SerialinkSimpleTest.CustomConstructor_1
Virtual Serial Name (slave): /dev/pts/1
Virtual serial port ready
[       OK ] SerialinkSimpleTest.CustomConstructor_1 (0 ms)
[ RUN      ] SerialinkSimpleTest.CustomConstructor_2
Virtual Serial Name (slave): /dev/pts/1
Virtual serial port ready
[       OK ] SerialinkSimpleTest.CustomConstructor_2 (0 ms)
....
....
....
[ RUN      ] SerialinkFramedDataTest.ReadTest_withPrefixAndSuffix_1
Virtual Serial Name (slave): /dev/pts/1
Virtual serial port ready
[       OK ] SerialinkFramedDataTest.ReadTest_withPrefixAndSuffix_1 (50 ms)
[ RUN      ] SerialinkFramedDataTest.ReadTest_withPrefixAndSuffix_2
Virtual Serial Name (slave): /dev/pts/1
Virtual serial port ready
[       OK ] SerialinkFramedDataTest.ReadTest_withPrefixAndSuffix_2 (50 ms)
[ RUN      ] SerialinkFramedDataTest.ReadTest_withPrefixAndSuffix_3
Virtual Serial Name (slave): /dev/pts/1
Virtual serial port ready
[       OK ] SerialinkFramedDataTest.ReadTest_withPrefixAndSuffix_3 (2767 ms)
[----------] 9 tests from SerialinkFramedDataTest (3021 ms total)
[----------] Global test environment tear-down
[==========] 35 tests from 2 test suites ran. (7739 ms total)
[  PASSED  ] 35 tests.
```

1.3 Using the Library

One way to use this library is by integrating it into your main application as a Git submodule. Here's an example of how to create a new project and integrate the Serialink library into it:

1. Create the project tree.

```
mkdir SerialReceiveApp
cd SerialReceiveApp
mkdir src
mkdir include
mkdir external
git init .
```

1. Add the Serialink library as an external library.

```
git submodule add https://github.com/JayaWikrama/Serialink.git external/Serialink
git submodule update --init --recursive
```

1. Create the main application source code.

```
nano src/main.cpp
```

Here is the sample source code.

```
#include <iostream>
#include <string.h>
#include "serialink.hpp"
const std::string errorMessage[] = {
    "Serial Port Has Not Been Opened",
    "Timeout",
    "Frame Format Has Not Been Setup"
};
void displayData(const std::vector <unsigned char> &data){
    for (auto i = data.begin(); i < data.end(); ++i){
        std::cout « std::hex « (int) *i;
        std::cout « " ";
    }
    std::cout « std::endl;
}
void crc16(DataFrame &frame, void *ptr) {
    /* Get Serialink object form function param */
    Serialink *obj = (Serialink *) ptr;
    /* Initialize crc16 param */
    unsigned short crc = 0x0000;
    unsigned short poly = 0x1021;
    /* Get data from Start Bytes until Data */
    std::vector <unsigned char> data = obj->getSpecificBufferAsVector(DataFrame::FRAME_TYPE_START_BYTES,
       DataFrame::FRAME_TYPE_DATA);
    std::cout « "Data from which the CRC value will be calculated:" « std::endl;
    displayData(data);
    /* Calculate crc16 */
    for (const auto &byte : data) {
        crc ^= (static_cast<unsigned short>(byte) « 8);
        for (int i = 0; i < 8; ++i) {
            if (crc & 0x8000) {
                crc = (crc « 1) ^ poly;
            } else {
                crc «= 1;
            }
        }
    }
    /* Compare received CRC with calculated CRC */
    unsigned short rcvCRC = 0;
    frame.getData((unsigned char *) &rcvCRC, 2);
    if (rcvCRC != crc){
        /* invalid crc -> trigger stop flag to __readFramedData__ method */
        obj->trigInvDataIndicator();
        std::cout « "CRC16 Invalid (0x" « std::hex « rcvCRC « " != 0x" « std::hex « crc « ")" « std::endl;
    }
}
void setupLengthByCommand(DataFrame &frame, void *ptr){
    int data = 0;
    /* Get Serialink object form function param */
    Serialink *obj = (Serialink *) ptr;
    /* Get DataFrame target */
    DataFrame *target = (*obj)[DataFrame::FRAME_TYPE_DATA];
    if (target == nullptr) return;
```

```
    frame.getData((unsigned char *) &data, 1);
    if (data == 0x35){
        /* setup 3 as data size of DataFrame::FRAME_TYPE_DATA */
        target->setSize(3);
    }
    else if (data == 0x36){
        /* setup 3 as data size of DataFrame::FRAME_TYPE_DATA */
        target->setSize(2);
    }
    else {
        /* invalid value found -> trigger stop flag to __readFramedData__ method */
        obj->trigInvDataIndicator();
    }
}
void serialSetupDataFrameProtocol(Serialink &serial){
    /* Frame Data Format / Protocol Example
     * | Start  Bytes |   Command   |  Main Data   | CRC16 Validator | Stop Bytes |
     * |:-------------|:------------|:-------------|:----------------|:-----------|
     * |    4 bytes   |   1 byte    |   N bytes    |     2 bytes     | 4 bytes    |
     * |  0x31323334  | 0x35 / 0x36 | based on Cmd |  init = 0x0000  | 0x39302D3D |
     *
     * If Command = 0x35, then the Main Data length is 3 bytes.
     * If Command = 0x36, then the Main Data length is 2 bytes.
     * If Command is not equal to 0x35 and 0x36, then the data is invalid.
     *
     * Use this input example: 3132333435363738159039302D3D
     */
    /* Configure Frame Format */
    DataFrame startBytes(DataFrame::FRAME_TYPE_START_BYTES, "1234");
    DataFrame cmdBytes(DataFrame::FRAME_TYPE_COMMAND, 1);
    /* Setup the handler function to determine the data length of DataFrame::FRAME_TYPE_DATA.
     * This function is called after all data from DataFrame::FRAME_TYPE_COMMAND is received.
     */
    cmdBytes.setPostExecuteFunction((const void *) &setupLengthByCommand, (void *) &serial);
    DataFrame dataBytes(DataFrame::FRAME_TYPE_DATA);
    DataFrame crcValidatorBytes(DataFrame::FRAME_TYPE_VALIDATOR, 2);
    /* Setup the handler function to validate Data by using crc16 validation.
     * This function is called after all data from DataFrame::FRAME_TYPE_VALIDATOR is received.
     */
    crcValidatorBytes.setPostExecuteFunction((const void *) &crc16, (void *) &serial);
    DataFrame stopBytes(DataFrame::FRAME_TYPE_STOP_BYTES, "90-=");
    /* Setup Frame Format to serial com */
    serial = startBytes + cmdBytes + dataBytes + crcValidatorBytes + stopBytes;
}
int main(int argc, char **argv){
    if (argc != 4){
        std::cout « "cmd: " « argv[0] « " <port> <timeout100ms> <keepAliveMs>" « std::endl;
        exit(0);
    }
    int ret = 0;
    std::vector <unsigned char> data;
    Serialink serial;
    /* Prepare Object */
    serial.setPort(argv[1]);
    serial.setBaudrate(B115200);
    serial.setTimeout(atoi(argv[2]));
    serial.setKeepAlive(atoi(argv[3]));
    serialSetupDataFrameProtocol(serial);
    /* Do serial communication */
    serial.openPort();
    while ((ret = serial.readFramedData()) != 0){
        if (ret == 4){
            std::cout « "Invalid Received Data Details:" « std::endl;
            if (serial.getBuffer(data) > 0){
                std::cout « "    Received Data: ";
                displayData(data);
            }
            if (serial.getRemainingBuffer(data) > 0){
                std::cout « "    Remaining Received Data: ";
                displayData(data);
            }
            std::cout « std::endl;
        }
        else {
            std::cout « errorMessage[ret - 1] « std::endl;
        }
    }
    serial.closePort();
    serial.getFormat()->getAllData(data);
    std::cout « "Received Success [" + std::to_string(serial.getDataSize()) + "]" « std::endl;
    std::cout « "    Data: ";
    displayData(data);
    return 0;
}
```

Save file by press `[Ctrl + o]` on your keyboard and then press `[Enter]`. After that press `[Ctrl + x]`.

1. Create a `CMakeLists.txt` file.

```
nano CMakeLists.txt
```

Here is `cmake` configuration:

```
cmake_minimum_required(VERSION 3.0.0)
set(PROJECT_VERSION_MAJOR "1")
set(PROJECT_VERSION_MINOR "1")
set(PROJECT_VERSION_PATCH "0")
project(SerialReceiver VERSION 1.1.0 LANGUAGES CXX)
include(FindPkgConfig)
# Set the default build type to Release if not specified
if(NOT CMAKE_BUILD_TYPE)
  set(CMAKE_BUILD_TYPE Release)
endif()
# Find and Check Library
find_package(PkgConfig REQUIRED)
# Specify the source files
set(SOURCE_FILES
    src/main.cpp
)
# Create the executable
add_executable(${PROJECT_NAME} ${SOURCE_FILES})
# Include directories
target_include_directories(${PROJECT_NAME} PUBLIC
  $<BUILD_INTERFACE:${CMAKE_CURRENT_SOURCE_DIR}/include>
  $<BUILD_INTERFACE:${CMAKE_CURRENT_SOURCE_DIR}/src>
  $<BUILD_INTERFACE:${CMAKE_CURRENT_SOURCE_DIR}/external/Serialink/include>
  $<INSTALL_INTERFACE:include>
)
# Ensure Serialink is built before the executable
add_dependencies(${PROJECT_NAME} Serialink-lib)
# Link the executable with the library
target_link_libraries(${PROJECT_NAME} PRIVATE Serialink-lib DataFrame-lib)
target_link_libraries(${PROJECT_NAME} PUBLIC -lpthread)
add_subdirectory(external/Serialink EXCLUDE_FROM_ALL)
set(USE_EXE_FUNC ON)
set(USE_POST_EXE_FUNC ON)
add_definitions(-D__USE_EXE_FUNC)
add_definitions(-D__USE_POST_FUNC)
# Set compiler and linker flags
set(CMAKE_CXX_FLAGS_DEBUG "${CMAKE_CXX_FLAGS_DEBUG} -g -O0")
set(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} -g -O0")
set(CMAKE_CXX_FLAGS_RELEASE "${CMAKE_CXX_FLAGS_RELEASE} -O3")
set(CMAKE_EXE_LINKER_FLAGS_RELEASE "${CMAKE_EXE_LINKER_FLAGS_RELEASE} -O3")
set(CMAKE_USE_RELATIVE_PATHS OFF)
```

Save file by press `[Ctrl + o]` on your keyboard and then press `[Enter]`. After that press `[Ctrl + x]`.

1.3.1 Explanation

- Line 29: The `$<BUILD_INTERFACE:${CMAKE_CURRENT_SOURCE_DIR}/external/Serialink/include>` command adds a reference to the header file location of the Serialink Library. This ensures that during the build process, CMake can locate the Serialink headers correctly.
- Line 34: The `add_dependencies(${PROJECT_NAME} Serialink-lib)` command specifies that the Serialink Library must be built before the main application. This ensures that the Serialink Library is compiled before the main project depends on it.
- Line 37: The `target_link_libraries(${PROJECT_NAME} PRIVATE Serialink-lib DataFrame-lib)` command links the Serialink Library and the DataFrame Library (a dependency of the Serialink Library) to the main application as private libraries. This means that these libraries are used internally by the main project.
- Line 40: The `add_subdirectory(external/Serialink EXCLUDE_FROM_ALL)` command adds the external/Serialink directory as part of the main project. The EXCLUDE_FROM_ALL option prevents CMake from building this subdirectory by default unless explicitly requested.

1. Build the project.

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

1. Run the application.

```
./SerialReceiver <port> <timeout100ms> <keepAliveMs>
```

Example:

```
./SerialReceiver /dev/ttyUSB0 10 500
```

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

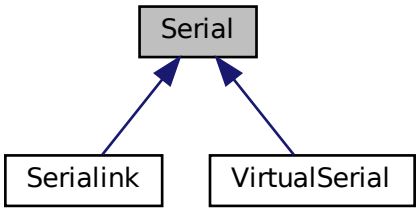
Chapter 5

Class Documentation

5.1 Serial Class Reference

`#include <serial.hpp>`

Inheritance diagram for Serial:



Public Member Functions

- Serial ()

  Default constructor.
- Serial (const std::string port, speed_t baud, unsigned int timeout)

  Custom constructor.
- Serial (const std::string port, speed_t baud, unsigned int timeout, unsigned int keepAliveMs)

  Custom constructor.
- ~Serial ()

  Destructor.
- void setPort (const std::string port)

  Sets the serial port device.
- void setBaudrate (speed_t baud)

  Sets the baud rate for communication.
- void setTimeout (unsigned int timeout)

Sets the communication timeout.

- void setKeepAlive (unsigned int keepAliveMs)

  Sets the keep-alive interval for communication.
- std::string getPort ()

  Gets the serial port.
- speed_t getBaudrate ()

  Gets the baud rate for serial communication.
- unsigned int getTimeout ()

  Gets the communication timeout.
- unsigned int getKeepAlive ()

  Gets the keep-alive interval for communication.
- int openPort ()

  Opens the serial port for communication.
- int readData (size_t sz, bool dontSplitRemainingData)

  Performs a serial data read operation.
- int readData (size_t sz)

  Overloaded method for `readData` to perform serial data reading.
- int readData ()

  Overloaded method for `readData` to perform serial data reading.
- int readStartBytes (const unsigned char *startBytes, size_t sz)

  Reads serial data until the desired start bytes are found.
- int readStartBytes (const char *startBytes)

  Overloaded method for `readStartBytes` with `const char*` input.
- int readStartBytes (const std::vector< unsigned char > startBytes)

  Overloaded method for `readStartBytes` with `std::vector<unsigned char>` input.
- int readStartBytes (const std::string startBytes)

  Overloaded method for `readStartBytes` with `std::string` input.
- int readUntilStopBytes (const unsigned char *stopBytes, size_t sz)

  Performs a serial data read operation until the specified stop bytes are detected.
- int readUntilStopBytes (const char *stopBytes)

  Overloaded function for `readUntilStopBytes` with input as `const char*`.
- int readUntilStopBytes (const std::vector< unsigned char > stopBytes)

  Overloaded function for `readUntilStopBytes` with input as `std::vector<unsigned char>`.
- int readUntilStopBytes (const std::string stopBytes)

  Overloaded function for `readUntilStopBytes` with input as `std::string`.
- int readStopBytes (const unsigned char *stopBytes, size_t sz)

  Reads serial data and checks if the data matches the specified stop bytes.
- int readStopBytes (const char *stopBytes)

  Function overloading for `readStopBytes` with input using `const char*`.
- int readStopBytes (const std::vector< unsigned char > stopBytes)

  Function overloading for `readStopBytes` with input using `std::vector`.
- int readStopBytes (const std::string stopBytes)

  Function overloading for `readStopBytes` with input using `std::string`.
- int readNBytes (size_t sz)

  Performs serial data reading until the desired amount of data is received.
- size_t getDataSize ()

  Retrieves the amount of successfully read data.
- size_t getBuffer (unsigned char *buffer, size_t maxBufferSz)

  Retrieves the read data buffer.
- size_t getBuffer (std::vector< unsigned char > &buffer)

  Overloaded method for `getBuffer` with `std::vector<unsigned char>` as the output parameter.

- std::vector< unsigned char > getBufferAsVector ()

  Retrieves the read data buffer as a vector.
- size_t getRemainingDataSize ()

  Retrieves the number of bytes in the remaining buffer.
- size_t getRemainingBuffer (unsigned char *buffer, size_t maxBufferSz)

  Retrieves the remaining serial data read outside of the data buffer.
- size_t getRemainingBuffer (std::vector< unsigned char > &buffer)

  *Overloading method for **getRemainingBuffer** with output parameter as vector.*
- std::vector< unsigned char > getRemainingBufferAsVector ()

  Retrieves remaining serial data that has been successfully read but is outside the main data buffer, returning it as a vector.
- int writeData (const unsigned char *buffer, size_t sz)

  Performs the operation of writing serial data.
- int writeData (const char *buffer)

  Method overloading of `writeData` with input as `const char*`.
- int writeData (const std::vector< unsigned char > buffer)

  Method overloading of `writeData` with input as `std::vector <unsigned char>`.
- int writeData (const std::string buffer)

  Method overloading of `writeData` with input as `std::string`.
- void closePort ()

  Closes the serial communication port.

Protected Member Functions

- void setFileDescriptor (HANDLE fd)

  Sets the file descriptor.
- HANDLE getFileDescriptor ()

  Gets the file descriptor.
- bool setupAttributes ()

  Configures serial port attributes.

Protected Attributes

- std::vector< unsigned char > data
- std::vector< unsigned char > remainingData

Private Attributes

- HANDLE fd
- speed_t baud
- unsigned int timeout
- unsigned int keepAliveMs
- std::string port
- pthread_mutex_t mtx
- pthread_mutex_t wmtx

5.1.1 Detailed Description

Definition at line 109 of file serial.hpp.

5.1.2 Constructor & Destructor Documentation

5.1.2.1 Serial() [1/3]

```
Serial::Serial ( )
```

Default constructor.

This constructor initializes private data members and parameters to their default values, including:

- `fd = -1` : File descriptor is set to an invalid state.
- `baud = B9600` : Baud rate is set to 9600 bps.
- `timeout = 10` : Timeout is set to 10 deciseconds (1 second).
- `keepAliveMs = 0` : Keep-alive interval is set to 0 milliseconds.
- `port = "/dev/ttyUSB0"` : Default serial port is set to `/dev/ttyUSB0`.
- Initializes the mutex for thread safety.

5.1.2.2 Serial() [2/3]

```
Serial::Serial (
            const std::string port,
            speed_t baud,
            unsigned int timeout )
```

Custom constructor.

This constructor initializes private data members to their default values (except for `port`, `baud`, and `timeout`), including:

- `fd = -1` : File descriptor is set to an invalid state.
- `keepAliveMs = 0` : Keep-alive interval is set to 0 milliseconds.
- Initializes the mutex for thread safety.

Parameters

port	The serial port device (e.g., "/dev/ttyUSB0").
baud	The baud rate for the serial communication.
timeout	Timeout value in units of 100 milliseconds (e.g., `10` equals 1 second).

5.1.2.3 Serial() [3/3]

```
Serial::Serial (
            const std::string port,
            speed_t baud,
            unsigned int timeout,
            unsigned int keepAliveMs )
```

Custom constructor.

This constructor initializes private data members to their default values (except for `port`, `baud`, `timeout`, and `keepAliveMs`), including:

- `fd = -1` : File descriptor is set to an invalid state.
- Initializes the mutex for thread safety.

Parameters

port	The serial port device (e.g., "/dev/ttyUSB0").
baud	The baud rate for the serial communication.
timeout	Timeout value in units of 100 milliseconds (e.g., `10` equals 1 second).
keepAliveMs	Keep-alive interval in milliseconds.

5.1.2.4 ~Serial()

```
Serial::~Serial ( )
```

Destructor.

This destructor is responsible for releasing any memory that has been allocated during the object's lifetime. It ensures that all allocated resources are properly freed, preventing memory leaks.

5.1.3 Member Function Documentation

5.1.3.1 closePort()

```
void Serial::closePort ( )
```

Closes the serial communication port.

This function is used to close the currently open serial communication port, ensuring that the port is no longer in use and that any associated system resources are released.

5.1.3.2 getBaudrate()

```
speed_t Serial::getBaudrate ( )
```

Gets the baud rate for serial communication.

This getter function retrieves the baud rate currently configured for serial communication.

Returns

The baud rate (e.g., `B9600` for 9600 bps).

5.1.3.3 getBuffer() [1/2]

```
size_t Serial::getBuffer (
            std::vector< unsigned char > & buffer )
```

Overloaded method for `getBuffer` with `std::vector<unsigned char>` as the output parameter.

Retrieves all the data that has been successfully read by the `read` method.

This overload allows you to use a `std::vector<unsigned char>` as the buffer to hold the serial data.

Parameters

buffer	A `std::vector<unsigned char>` to hold the serial data that has been successfully read.

Returns

The size of the serial data read.

5.1.3.4 getBuffer() [2/2]

```
size_t Serial::getBuffer (
            unsigned char * buffer,
            size_t maxBufferSz )
```

Retrieves the read data buffer.

This function retrieves all the data that has been successfully read by the `read` method.

Parameters

buffer	A variable to hold the serial data that has been successfully read.
maxBufferSz	The maximum size of the data that can be accommodated in the buffer.

Returns

The size of the serial data read.

5.1.3.5 getBufferAsVector()

```
std::vector<unsigned char> Serial::getBufferAsVector ( )
```

Retrieves the read data buffer as a vector.

This method returns all the data that has been successfully read by the `read` method as a `std::vector<unsigned char>`.

Returns

A `std::vector<unsigned char>` containing the serial data that has been successfully read.

5.1.3.6 getDataSize()

```
size_t Serial::getDataSize ( )
```

Retrieves the amount of successfully read data.

This function retrieves the information about the size of the data that has been successfully read.

Returns

The size of the serial data in bytes.

5.1.3.7 getFileDescriptor()

```
HANDLE Serial::getFileDescriptor ( )  [protected]
```

Gets the file descriptor.

This getter function retrieves the current value of the file descriptor.

Returns

The current file descriptor.

5.1.3.8 getKeepAlive()

```
unsigned int Serial::getKeepAlive ( )
```

Gets the keep-alive interval for communication.

This getter function retrieves the maximum wait time configured for receiving the next byte of serial data after the initial byte has been successfully received. The interval is specified in milliseconds.

Returns

The keep-alive interval in milliseconds.

5.1.3.9 getPort()

```
std::string Serial::getPort ( )
```

Gets the serial port.

This getter function retrieves the serial port currently being used.

Returns

The serial port as a string (e.g., "/dev/ttyUSB0").

5.1.3.10 getRemainingBuffer() [1/2]

```
size_t Serial::getRemainingBuffer (
            std::vector< unsigned char > & buffer )
```

Overloading method for **getRemainingBuffer** with output parameter as vector.

Retrieves all remaining serial data that has been successfully read but is outside the main data buffer.

Parameters

buffer	Variable to hold the remaining serial data read, provided as a vector.

Returns

The size of the remaining serial data read.

5.1.3.11 getRemainingBuffer() [2/2]

```
size_t Serial::getRemainingBuffer (
            unsigned char * buffer,
            size_t maxBufferSz )
```

Retrieves the remaining serial data read outside of the data buffer.

This method extracts all remaining serial data that has been successfully read but is outside the main data buffer.

Parameters

buffer	Variable to hold the remaining serial data read.
maxBufferSz	Maximum size of data that can be held by the buffer variable.

Returns

The size of the serial data read.

5.1.3.12 getRemainingBufferAsVector()

```
std::vector<unsigned char> Serial::getRemainingBufferAsVector ( )
```

Retrieves remaining serial data that has been successfully read but is outside the main data buffer, returning it as a vector.

This method retrieves all remaining serial data that has been read successfully but is not included in the main data buffer, and returns it as a vector.

Returns

std::vector<unsigned char> containing the remaining serial data that has been successfully read.

5.1.3.13 getRemainingDataSize()

```
size_t Serial::getRemainingDataSize ( )
```

Retrieves the number of bytes in the remaining buffer.

This method provides the size of the data that is still present in the remaining buffer.

Returns

The size of the remaining data in bytes.

5.1.3.14 getTimeout()

```
unsigned int Serial::getTimeout ( )
```

Gets the communication timeout.

This getter function retrieves the timeout value configured for serial communication. The timeout is specified in units of 100 milliseconds.

Returns

The timeout value (e.g., `10` for a 1-second timeout).

5.1.3.15 openPort()

```
int Serial::openPort ( )
```

Opens the serial port for communication.

This function attempts to open the specified serial port for communication. It configures the port according to the current settings and prepares it for data transfer.

Returns

0 if the port is successfully opened.

1 if the port fails to open.

5.1.3.16 readData() [1/3]

```
int Serial::readData ( )
```

Overloaded method for `readData` to perform serial data reading.

This overloaded method performs a serial data read operation. The successfully read serial data can be accessed using the `Serial::getBuffer` method.

Returns

`0` if the operation is successful.

`1` if the port is not open.

`2` if a timeout occurs.

5.1.3.17 readData() [2/3]

```
int Serial::readData (
            size_t sz )
```

Overloaded method for `readData` to perform serial data reading.

This overloaded method performs a serial data read operation. The successfully read serial data can be accessed using the `Serial::getBuffer` method.

Parameters

sz	The number of bytes to read. A value of `0` means that the read operation is unlimited (up to the `keepAliveMs` timeout).

Returns

`0` if the operation is successful.

`1` if the port is not open.

`2` if a timeout occurs.

5.1.3.18 readData() [3/3]

```
int Serial::readData (
            size_t sz,
            bool dontSplitRemainingData )
```

Performs a serial data read operation.

This function reads data from the serial port without separating the successfully read data into the desired size and remaining data. The read serial data can be accessed using the `Serial::getBuffer` method.

Parameters

sz	The number of bytes to read. A value of `0` means that the read operation is unlimited (up to the `keepAliveMs` timeout).
dontSplitRemainingData	A flag to disable automatic data splitting based on the amount of data requested.

Returns

`0` if the operation is successful.

`1` if the port is not open.

`2` if a timeout occurs.

5.1.3.19 readNBytes()

```
int Serial::readNBytes (
            size_t sz )
```

Performs serial data reading until the desired amount of data is received.

This function performs the operation of reading serial data until the specified amount of data is fulfilled. The operation retries up to 3 times, starting from the first data received. The read serial data can be accessed using the `Serial::getBuffer` method.

Parameters

sz	The size of the serial data to be read.

Returns

`0` if successful.

`1` if the port is not open.

`2` if a timeout occurs.

5.1.3.20 readStartBytes() [1/4]

```
int Serial::readStartBytes (
            const char * startBytes )
```

Overloaded method for `readStartBytes` with `const char*` input.

This overloaded function performs a serial data read operation until the specified start bytes are detected. Any serial data read before the start bytes are found is automatically discarded. The read serial data can be accessed using the `Serial::getBuffer` method.

Parameters

startBytes	A pointer to the start bytes data to be detected, provided as a `const char*`.

Returns

`0` if the operation is successful.

`1` if the port is not open.

`2` if a timeout occurs.

5.1.3.21 readStartBytes() [2/4]

```
int Serial::readStartBytes (
            const std::string startBytes )
```

Overloaded method for `readStartBytes` with `std::string` input.

This overloaded function performs a serial data read operation until the specified start bytes are detected. Any serial data read before the start bytes are found is automatically discarded. The read serial data can be accessed using the `Serial::getBuffer` method.

Parameters

startBytes	A string containing the start bytes data to be detected.

Returns

0 if the operation is successful.

1 if the port is not open.

2 if a timeout occurs.

5.1.3.22 readStartBytes() [3/4]

```
int Serial::readStartBytes (
            const std::vector< unsigned char > startBytes )
```

Overloaded method for `readStartBytes` with `std::vector<unsigned char>` input.

This overloaded function performs a serial data read operation until the specified start bytes are detected. Any serial data read before the start bytes are found is automatically discarded. The read serial data can be accessed using the `Serial::getBuffer` method.

Parameters

startBytes	A vector containing the start bytes data to be detected.

Returns

0 if the operation is successful.

1 if the port is not open.

2 if a timeout occurs.

5.1.3.23 readStartBytes() [4/4]

```
int Serial::readStartBytes (
            const unsigned char * startBytes,
            size_t sz )
```

Reads serial data until the desired start bytes are found.

This function performs a serial data read operation until the specified start bytes are detected. Any serial data read before the start bytes are found is automatically discarded. The read serial data can be accessed using the `Serial::getBuffer` method.

Parameters

startBytes	A pointer to the start bytes data to be detected.
sz	The size of the start bytes data to be detected.

Returns

0 if the operation is successful.

1 if the port is not open.

2 if a timeout occurs.

5.1.3.24 readStopBytes() [1/4]

```
int Serial::readStopBytes (
            const char * stopBytes )
```

Function overloading for `readStopBytes` with input using `const char*`.

This method performs serial data reading while simultaneously checking if the data matches the specified stop bytes. The read serial data can be accessed using the `Serial::getBuffer` method.

Parameters

stopBytes	A C-style string (null-terminated) representing the stop bytes to be detected.

Returns

0 if the operation is successful and the data is valid.

1 if the port is not open.

2 if a timeout occurs.

3 if data is read but does not match the specified stop bytes.

5.1.3.25 readStopBytes() [2/4]

```
int Serial::readStopBytes (
            const std::string stopBytes )
```

Function overloading for `readStopBytes` with input using `std::string`.

This method performs serial data reading while simultaneously checking if the data matches the specified stop bytes. The read serial data can be accessed using the `Serial::getBuffer` method.

Parameters

stopBytes	A `std::string` object representing the stop bytes to be detected.

Returns

0 if the operation is successful and the data is valid.

1 if the port is not open.

2 if a timeout occurs.

3 if data is read but does not match the specified stop bytes.

5.1.3.26 readStopBytes() [3/4]

```
int Serial::readStopBytes (
            const std::vector< unsigned char > stopBytes )
```

Function overloading for `readStopBytes` with input using `std::vector`.

This method performs serial data reading while simultaneously checking if the data matches the specified stop bytes. The read serial data can be accessed using the `Serial::getBuffer` method.

Parameters

stopBytes	A `std::vector` containing the stop bytes data to be detected.

Returns

0 if the operation is successful and the data is valid.

1 if the port is not open.

2 if a timeout occurs.

3 if data is read but does not match the specified stop bytes.

5.1.3.27 readStopBytes() [4/4]

```
int Serial::readStopBytes (
            const unsigned char * stopBytes,
            size_t sz )
```

Reads serial data and checks if the data matches the specified stop bytes.

This method performs serial data reading while simultaneously checking if the data matches the desired stop bytes. The read serial data can be accessed using the `Serial::getBuffer` method.

Parameters

stopBytes	A pointer to the stop bytes data to be detected.
sz	The size of the stop bytes data to be detected.

Returns

0 if the operation is successful and the data is valid.

1 if the port is not open.

2 if a timeout occurs.

3 if data is read but does not match the specified stop bytes.

5.1.3.28 readUntilStopBytes() [1/4]

```
int Serial::readUntilStopBytes (
            const char * stopBytes )
```

Overloaded function for `readUntilStopBytes` with input as `const char*`.

This function reads serial data until the specified stop bytes are detected. Any serial data read up to and including the stop bytes is automatically stored in the buffer. The read serial data can be accessed using the `Serial::getBuffer` method.

Parameters

stopBytes	A pointer to a null-terminated character array representing the stop bytes to be detected.

Returns

0 if the operation is successful.

1 if the port is not open.

2 if a timeout occurs.

5.1.3.29 readUntilStopBytes() [2/4]

```
int Serial::readUntilStopBytes (
            const std::string stopBytes )
```

Overloaded function for `readUntilStopBytes` with input as `std::string`.

This function reads serial data until the specified stop bytes are detected. Any serial data read up to and including the stop bytes is automatically stored in the buffer. The read serial data can be accessed using the `Serial::getBuffer` method.

Parameters

stopBytes	A string representing the stop bytes to be detected.

Returns

0 if the operation is successful.

1 if the port is not open.

2 if a timeout occurs.

5.1.3.30 readUntilStopBytes() [3/4]

```
int Serial::readUntilStopBytes (
            const std::vector< unsigned char > stopBytes )
```

Overloaded function for `readUntilStopBytes` with input as `std::vector<unsigned char>`.

This function reads serial data until the specified stop bytes are detected. Any serial data read up to and including the stop bytes is automatically stored in the buffer. The read serial data can be accessed using the `Serial::getBuffer` method.

Parameters

stopBytes	A vector of `unsigned char` representing the stop bytes to be detected.

Returns

`0` if the operation is successful.

`1` if the port is not open.

`2` if a timeout occurs.

5.1.3.31 readUntilStopBytes() [4/4]

```
int Serial::readUntilStopBytes (
            const unsigned char * stopBytes,
            size_t sz )
```

Performs a serial data read operation until the specified stop bytes are detected.

This function reads serial data until the specified stop bytes are detected. Any serial data read up to and including the stop bytes is automatically stored in the buffer. The read serial data can be accessed using the `Serial::getBuffer` method.

Parameters

stopBytes	The data representing the stop bytes to be detected.
sz	The size of the stop bytes data to be detected.

Returns

`0` if the operation is successful.

`1` if the port is not open.

`2` if a timeout occurs.

5.1.3.32 setBaudrate()

```
void Serial::setBaudrate (
            speed_t baud )
```

Sets the baud rate for communication.

This setter function configures the baud rate used for serial communication.

Parameters

baud	The baud rate (e.g., `B9600` for 9600 bps).

5.1.3.33 setFileDescriptor()

```
void Serial::setFileDescriptor (
            HANDLE fd )  [protected]
```

Sets the file descriptor.

This setter function assigns a value to the file descriptor.

Parameters

fd	The file descriptor to be set.

5.1.3.34 setKeepAlive()

```
void Serial::setKeepAlive (
            unsigned int keepAliveMs )
```

Sets the keep-alive interval for communication.

This setter function configures the maximum wait time for receiving the next byte of serial data after the initial byte has been received. This helps maintain the connection by ensuring timely data reception.

Parameters

keepAliveMs	The keep-alive interval in milliseconds.

5.1.3.35 setPort()

```
void Serial::setPort (
```

```
            const std::string port )
```

Sets the serial port device.

This setter function configures the serial port device to be used for communication.

Parameters

port	The serial port device (e.g., "/dev/ttyUSB0").

5.1.3.36 setTimeout()

```
void Serial::setTimeout (
            unsigned int timeout )
```

Sets the communication timeout.

This setter function configures the timeout for serial communication. The timeout value is specified in units of 100 milliseconds.

Parameters

timeout	The timeout value (e.g., `10` for a 1-second timeout).

5.1.3.37 setupAttributes()

```
bool Serial::setupAttributes ( )  [protected]
```

Configures serial port attributes.

This function sets up or configures the attributes of the file descriptor for a successfully opened serial port.

Returns

true if the configuration is successful.

false if the configuration fails.

5.1.3.38 writeData() [1/4]

```
int Serial::writeData (
            const char * buffer )
```

Method overloading of `writeData` with input as `const char*`.

This method writes the specified data to the serial port. This overload allows the data to be passed as a `const char*`.

Parameters

buffer	Data to be written.

Returns

0 if the operation is successful.

1 if the port is not open.

2 if the data write operation fails.

5.1.3.39 writeData() [2/4]

```
int Serial::writeData (
            const std::string buffer )
```

Method overloading of `writeData` with input as `std::string`.

This method writes the specified data to the serial port. This overload allows the data to be passed as a `std::string`.

Parameters

buffer	Data to be written.

Returns

0 if the operation is successful.

1 if the port is not open.

2 if the data write operation fails.

5.1.3.40 writeData() [3/4]

```
int Serial::writeData (
            const std::vector< unsigned char > buffer )
```

Method overloading of `writeData` with input as `std::vector <unsigned char>`.

This method writes the specified data to the serial port. This overload allows the data to be passed as a `std::vector <unsigned char>`.

Parameters

buffer	Data to be written.

Returns

0 if the operation is successful.

1 if the port is not open.

2 if the data write operation fails.

5.1.3.41 writeData() [4/4]

```
int Serial::writeData (
            const unsigned char * buffer,
            size_t sz )
```

Performs the operation of writing serial data.

This method writes the specified data to the serial port.

Parameters

buffer	Data to be written.
sz	Size of the data to be written.

Returns

0 if the operation is successful.

1 if the port is not open.

2 if the data write operation fails.

5.1.4 Member Data Documentation

5.1.4.1 baud

```
speed_t Serial::baud  [private]
```

Definition at line 116 of file serial.hpp.

5.1.4.2 data

```
std::vector<unsigned char> Serial::data  [protected]
```

Definition at line 123 of file serial.hpp.

5.1.4.3 fd

```
HANDLE Serial::fd  [private]
```

Definition at line 114 of file serial.hpp.

5.1.4.4 keepAliveMs

```
unsigned int Serial::keepAliveMs  [private]
```

Definition at line 118 of file serial.hpp.

5.1.4.5 mtx

```
pthread_mutex_t Serial::mtx  [private]
```

Definition at line 120 of file serial.hpp.

5.1.4.6 port

```
std::string Serial::port  [private]
```

Definition at line 119 of file serial.hpp.

5.1.4.7 remainingData

```
std::vector<unsigned char> Serial::remainingData  [protected]
```

Definition at line 124 of file serial.hpp.

5.1.4.8 timeout

```
unsigned int Serial::timeout  [private]
```

Definition at line 117 of file serial.hpp.

5.1.4.9 wmtx

```
pthread_mutex_t Serial::wmtx  [private]
```

Definition at line 121 of file serial.hpp.

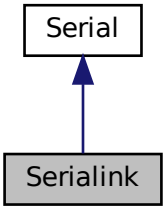
The documentation for this class was generated from the following file:

- Serialink/include/serial.hpp

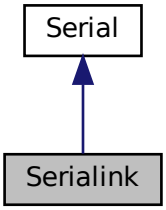
5.2 Serialink Class Reference

```
#include <serialink.hpp>
```

Inheritance diagram for Serialink:



Collaboration diagram for Serialink:



Public Member Functions

- Serialink ()

  Default constructor.
- ~Serialink ()

  Destructor.
- DataFrame * getFormat ()

  Retrieves the memory address of the frame format.
- void trigInvDataIndicator ()

  Stops reading framed serial data.
- int readFramedData ()

  Performs serial data read operations with a custom frame format.
- int writeFramedData ()

  Performs serial data write operations with a custom frame format.
- std::vector< unsigned char > getSpecificBufferAsVector (DataFrame::FRAME_TYPE_t begin, DataFrame::FRAME_TYPE_t end)

  Retrieves a buffer of data read and stored in the Framed Data within a specified range.
- std::vector< unsigned char > getSpecificBufferAsVector (const DataFrame *begin, const DataFrame *end)

  *Overloaded method of **getSpecificBufferAsVector**.*
- Serialink & operator= (const DataFrame &obj)
- Serialink & operator+= (const DataFrame &obj)
- Serialink & operator+ (const DataFrame &obj)
- DataFrame * operator[] (int idx)
- DataFrame * operator[] (DataFrame::FRAME_TYPE_t type)
- DataFrame * operator[] (std::pair< DataFrame::FRAME_TYPE_t, int > params)

Private Attributes

- bool isFormatValid
- DataFrame * frameFormat

Additional Inherited Members

5.2.1 Detailed Description

Definition at line 43 of file serialink.hpp.

5.2.2 Constructor & Destructor Documentation

5.2.2.1 Serialink()

```
Serialink::Serialink ( )
```

Default constructor.

Initializes private data members and parameters to their default values.

5.2.2.2 ~Serialink()

```
Serialink::~Serialink ( )
```

Destructor.

Releases any allocated memory.

5.2.3 Member Function Documentation

5.2.3.1 getFormat()

```
DataFrame* Serialink::getFormat ( )
```

Retrieves the memory address of the frame format.

This function returns the address of the `frameFormat` data member.

Returns

The memory address of the `frameFormat`.

5.2.3.2 getSpecificBufferAsVector() [1/2]

```
std::vector<unsigned char> Serialink::getSpecificBufferAsVector (
            const DataFrame * begin,
            const DataFrame * end )
```

Overloaded method of **getSpecificBufferAsVector**.

This method performs the same data extraction operation as the other overload but is designed to handle cases where duplicate Frame Formats exist within the Framed Data. It retrieves data from the buffer that has been successfully read and stored within the specified range.

Parameters

begin	Pointer to the starting point for data extraction.
end	Pointer to the ending point for data extraction.

Returns

A vector containing the data.

5.2.3.3 getSpecificBufferAsVector() [2/2]

```
std::vector<unsigned char> Serialink::getSpecificBufferAsVector (
            DataFrame::FRAME_TYPE_t begin,
            DataFrame::FRAME_TYPE_t end )
```

Retrieves a buffer of data read and stored in the Framed Data within a specified range.

This method extracts data from the buffer that has been successfully read and stored in the Framed Data within the specified range. This version is suitable for data with unique Frame Formats in each frame (no duplicate Frame Types).

Parameters

begin	Reference to the starting point for data extraction.
end	Reference to the ending point for data extraction.

Returns

A vector containing the data.

5.2.3.4 operator+()

```
Serialink& Serialink::operator+ (
            const DataFrame & obj )
```

5.2.3.5 operator+=()

```
Serialink& Serialink::operator+= (
            const DataFrame & obj )
```

5.2.3.6 operator=()

```
Serialink& Serialink::operator= (
            const DataFrame & obj )
```

5.2.3.7 operator[]() [1/3]

```
DataFrame* Serialink::operator[] (
            DataFrame::FRAME_TYPE_t type )
```

5.2.3.8 operator[]() [2/3]

```
DataFrame* Serialink::operator[] (
            int idx )
```

5.2.3.9 operator[]() [3/3]

```
DataFrame* Serialink::operator[] (
            std::pair< DataFrame::FRAME_TYPE_t, int > params )
```

5.2.3.10 readFramedData()

```
int Serialink::readFramedData ( )
```

Performs serial data read operations with a custom frame format.

This function executes serial data reading operations using a specific frame format. The read serial data can be retrieved using the __Serial::getBuffer__ method.

Returns

0 on success.

1 if the port is not open.

2 if a timeout occurs.

3 if the frame format is not set up.

4 if the frame data format is invalid.

5.2.3.11 trigInvDataIndicator()

```
void Serialink::trigInvDataIndicator ( )
```

Stops reading framed serial data.

This function sets up variables that act as indicators for the validity of the received serial data, allowing it to stop reading framed serial data from user space. It achieves this by using a post-execution function configured within the DataFrame class.

5.2.3.12 writeFramedData()

```
int Serialink::writeFramedData ( )
```

Performs serial data write operations with a custom frame format.

This function executes serial data write operations using a specific frame format.

Returns

0 on success.

1 if the port is not open.

2 if a timeout occurs.

3 if there is no data to write.

5.2.4 Member Data Documentation

5.2.4.1 frameFormat

```
DataFrame* Serialink::frameFormat  [private]
```

Definition at line 46 of file serialink.hpp.

5.2.4.2 isFormatValid

```
bool Serialink::isFormatValid  [private]
```

Definition at line 45 of file serialink.hpp.

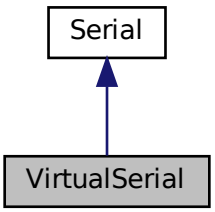
The documentation for this class was generated from the following file:

- Serialink/include/serialink.hpp

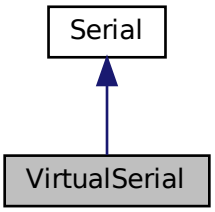
5.3 VirtualSerial Class Reference

`#include <virtuser.hpp>`

Inheritance diagram for VirtualSerial:



Collaboration diagram for VirtualSerial:



Public Member Functions

- VirtualSerial ()

  Default constructor.
- VirtualSerial (speed_t baud, unsigned int timeout, unsigned int keepAliveMs)

  Custom constructor.
- ~VirtualSerial ()

  Destructor.
- void setCallback (const void *func, void *param)

  Sets the callback function for read and write operations on the master port of the virtual serial port.
- const void * getCallbackFunction ()

  Gets the callback function used for read and write operations on the master port of the virtual serial port.
- void * getCallbackParam ()

  Gets the parameter pointer used for the callback function in read and write operations on the master port of the virtual serial port.
- std::string getVirtualPortName ()

  Gets the name of the virtual serial port.
- bool begin ()

  Method to start executing the callback function.

Private Attributes

- std::string virtualPortName
- const void * callbackFunc
- void * callbackParam

Additional Inherited Members

5.3.1 Detailed Description

Definition at line 59 of file virtuser.hpp.

5.3.2 Constructor & Destructor Documentation

5.3.2.1 VirtualSerial() [1/2]

```
VirtualSerial::VirtualSerial ( )
```

Default constructor.

Initializes a virtual serial port. The name or address of the virtual port for the slave can be obtained using the `getVirtualPortName` method.

Default parameters:

- Baud rate: B9600
- Timeout: 10 (1 second)
- Keep-alive interval: 0 milliseconds

Initializes a mutex for thread safety.

5.3.2.2 VirtualSerial() [2/2]

```
VirtualSerial::VirtualSerial (
            speed_t baud,
            unsigned int timeout,
            unsigned int keepAliveMs )
```

Custom constructor.

Initializes a virtual serial port with specified settings. The name or address of the virtual port for the slave can be obtained using the `getVirtualPortName` method.

Parameters

baud	The baud rate for the serial communication.
timeout	The timeout period in 100 milliseconds.
keepAliveMs	The keep-alive interval in milliseconds.

5.3.2.3 ~VirtualSerial()

```
VirtualSerial::~VirtualSerial ( )
```

Destructor.

Releases all allocated memory and closes the master port of the virtual serial port.

5.3.3 Member Function Documentation

5.3.3.1 begin()

```
bool VirtualSerial::begin ( )
```

Method to start executing the callback function.

This method initiates the execution of the previously set callback function.

Returns

`false` if the callback function has not been set up.

`true` if the callback function has been successfully executed.

5.3.3.2 getCallbackFunction()

```
const void* VirtualSerial::getCallbackFunction ( )
```

Gets the callback function used for read and write operations on the master port of the virtual serial port.

Returns

Pointer to the callback function.

5.3.3.3 getCallbackParam()

```
void* VirtualSerial::getCallbackParam ( )
```

Gets the parameter pointer used for the callback function in read and write operations on the master port of the virtual serial port.

Returns

Pointer to the parameter of the callback function.

5.3.3.4 getVirtualPortName()

```
std::string VirtualSerial::getVirtualPortName ( )
```

Gets the name of the virtual serial port.

Returns

The name of the virtual serial port (slave).

5.3.3.5 setCallback()

```
void VirtualSerial::setCallback (
            const void * func,
            void * param )
```

Sets the callback function for read and write operations on the master port of the virtual serial port.

This method allows you to specify a callback function that will be used for handling data operations (reading and writing) on the virtual serial port's master port.

Parameters

func	Pointer to the callback function.
param	Pointer to the parameter for the callback function.

5.3.4 Member Data Documentation

5.3.4.1 callbackFunc

```
const void* VirtualSerial::callbackFunc  [private]
```

Definition at line 62 of file virtuser.hpp.

5.3.4.2 callbackParam

```
void* VirtualSerial::callbackParam  [private]
```

Definition at line 63 of file virtuser.hpp.

5.3.4.3 virtualPortName

```
std::string VirtualSerial::virtualPortName  [private]
```

Definition at line 61 of file virtuser.hpp.

The documentation for this class was generated from the following file:

- Serialink/include/virtuser.hpp

Chapter 6

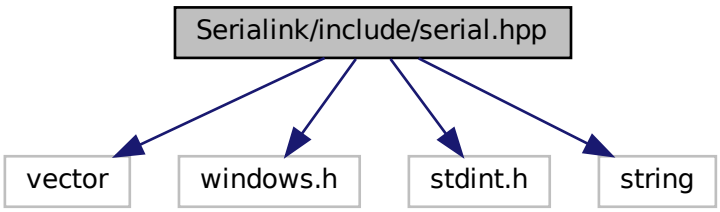
File Documentation

6.1 Serialink/include/serial.hpp File Reference

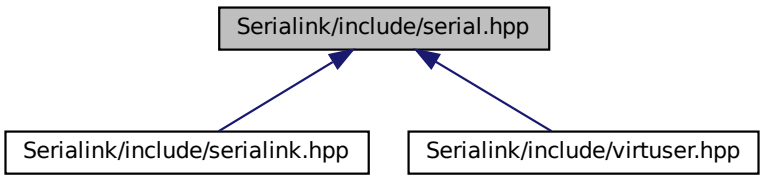
Enhanced Serial Communication Functions.

```
#include <vector>
#include <windows.h>
#include <stdint.h>
#include <string>
```

Include dependency graph for serial.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class Serial

Typedefs

- typedef int pthread_mutex_t
- typedef enum _speed_t speed_t

Enumerations

- enum _speed_t {
  B0 = 0 , B50 = 50 , B75 = 75 , B110 = 110 ,
  B134 = 134 , B150 = 150 , B200 = 200 , B300 = 300 ,
  B600 = 600 , B1200 = 1200 , B1800 = 1800 , B2400 = 2400 ,
  B4800 = 4800 , B9600 = 9600 , B19200 = 19200 , B38400 = 38400 ,
  B57600 = 57600 , B115200 = 115200 , B230400 = 230400 , B460800 = 460800 ,
  B192600 = 192600 }

Functions

- int pthread_mutex_init (pthread_mutex_t ∗mtx, void ∗ptr)
- int pthread_mutex_lock (pthread_mutex_t ∗mtx)
- int pthread_mutex_unlock (pthread_mutex_t ∗mtx)

6.1.1 Detailed Description

Enhanced Serial Communication Functions.

This file contains a collection of functions and commands designed to facilitate and extend basic serial communication in C++. These functions are intended to simplify the process of setting up, sending, receiving, and managing data over serial connections. The enhancements provided in this header file go beyond the standard library functions, offering more flexibility and control for developers working with serial interfaces.

The key functionalities include:

- Initialization and configuration of serial ports.
- Sending and receiving data over serial connections.
- Error handling and diagnostics for serial communication.
- Utility functions for managing serial buffers and flow control.

The functions in this file are designed to be easy to integrate into various projects, providing a robust foundation for serial communication in embedded systems, networking, or any application that requires serial data transmission.

Note

This file is a part of a larger project focusing on enhancing serial communication capabilities in C++ applications.

Version

1.0.0

Date

2024-08-25

Author

Jaya Wikrama

6.1.2 Typedef Documentation

6.1.2.1 pthread_mutex_t

```
typedef int pthread_mutex_t
```

Note

On Windows, this implementation is created without using Mutex and thread functionality.

Definition at line 68 of file serial.hpp.

6.1.2.2 speed_t

```
typedef enum _speed_t speed_t
```

6.1.3 Enumeration Type Documentation

6.1.3.1 _speed_t

```
enum _speed_t
```

Enumerator

B0	
B50	
B75	
B110	
B134	
B150	
B200	
B300	
B600	
B1200	
B1800	
B2400	
B4800	
B9600	
B19200	
B38400	
B57600	
B115200	
B230400	
B460800	
B192600	

Definition at line 82 of file serial.hpp.

```
82                          {
83   B0 = 0,
84   B50 = 50,
85   B75 = 75,
86   B110 = 110,
87   B134 = 134,
88   B150 = 150,
89   B200 = 200,
90   B300 = 300,
91   B600 = 600,
92   B1200 = 1200,
93   B1800 = 1800,
94   B2400 = 2400,
95   B4800 = 4800,
96   B9600 = 9600,
97   B19200 = 19200,
98   B38400 = 38400,
99   B57600 = 57600,
100   B115200 = 115200,
101   B230400 = 230400,
102   B460800 = 460800,
103   B192600 = 192600
104 } speed_t;
```

6.1.4 Function Documentation

6.1.4.1 pthread_mutex_init()

```
int pthread_mutex_init (
            pthread_mutex_t * mtx,
            void * ptr )
```

Definition at line 69 of file serial.hpp.

```
69                                                      {
70   // do noting
71   return 0;
72 }
```

6.1.4.2 pthread_mutex_lock()

```
int pthread_mutex_lock (
            pthread_mutex_t * mtx )
```

Definition at line 73 of file serial.hpp.

```
73                                              {
74   // do noting
75   return 0;
76 }
```

6.1.4.3 pthread_mutex_unlock()

```
int pthread_mutex_unlock (
            pthread_mutex_t * mtx )
```

Definition at line 77 of file serial.hpp.

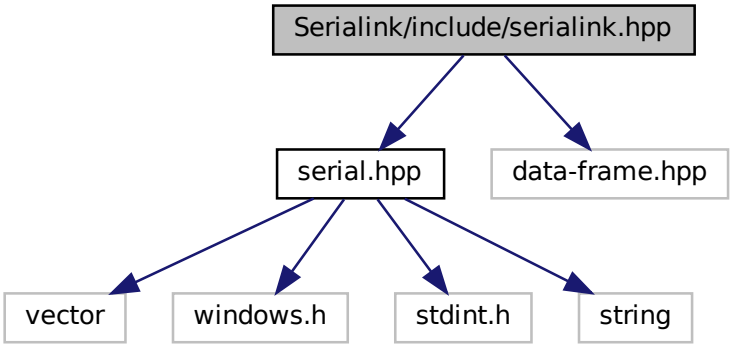
```
77                                                {
78   // do noting
79   return 0;
80 }
```

6.2 Serialink/include/serialink.hpp File Reference

Advance Serial Protocol (with framed data) library.

```
#include "serial.hpp"
#include "data-frame.hpp"
```

Include dependency graph for serialink.hpp:



Classes

- class Serialink

6.2.1 Detailed Description

Advance Serial Protocol (with framed data) library.

This library provides a robust implementation of a serial communication protocol that includes data framing. It is designed for reliable data exchange over serial connections, supporting error detection and packet-based communication.

Version

1.0.0

Date

2024-08-27

Author

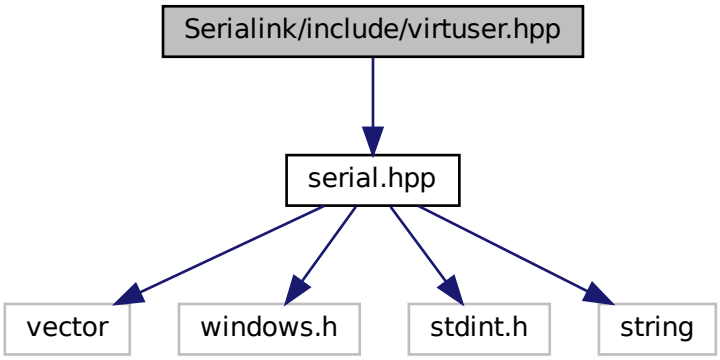
Jaya Wikrama

6.3 Serialink/include/virtuser.hpp File Reference

Header file for virtual serial port functionality and handler functions.

```
#include "serial.hpp"
```

Include dependency graph for virtuser.hpp:



Classes

- class VirtualSerial

6.3.1 Detailed Description

Header file for virtual serial port functionality and handler functions.

This file defines a set of functions and classes for creating and managing virtual serial ports. The functionalities provided allow for the simulation of serial communication interfaces, which can be useful for testing, debugging, or developing applications that interact with serial ports without requiring physical hardware.

The key components included in this file are:

- Definitions and classes for virtual serial ports.
- Functions to create, configure, and manage virtual serial ports.
- Handler functions to manage data flow and events on virtual serial ports.
- Utility functions for simulating serial communication scenarios and testing.

The virtual serial ports provided by this file can be used to simulate communication between software components or to test applications that rely on serial data exchange. The handler functions offer control over how data is processed and how events are handled, providing a flexible and powerful toolset for developers.

Note

This file is intended for use in environments where physical serial ports are not available or practical. It is particularly useful in development and testing scenarios.

Version

1.0.0

Date

2024-07-29

Author

Jaya Wikrama

6.4 Serialink/README.md File Reference

Index