UdpSocket C++ library

v3.1.2

Table of contents

Overview

UdpSocket C++ library provides methods to work with UDP port (open, close, send data and receive data). **UdpSocket** library is cross-platform and compatible with Windows and Linux OS. Main file **UdpSocket.h** includes declaration of **UdpSocket** class which provides methods to work with UDP socket. The library requires C++17 standard. The library doesn't have any third-party dependency. The library is licensed under the **Apache 2.0** license.

Versions

Table 1 - Library versions.

Version	Release date	What's new
1.0.0	10.10.2021	First version
2.0.0	17.05.2022	- Class interface changed. - Added Tracker library to print debug info.
3.0.0	13.05.2023	- Class interface changed. - Tracer library excluded.
3.0.1	13.05.2023	- Fixed send data to external address.
3.1.0	27.08.2023	- License added. - Repository made public.
3.1.1	26.03.2024	- Documentation updated.
3.1.2	22.05.2024	- Documentation updated.

UdpSocket class description

UdpSocket class declaration

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

```
class UdpSocket
{
public:

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

    /// Class constructor.
    UdpSocket();

    /// Class destructor.
    ~UdpSocket();

    /// Open UDP socket.
    bool open(uint16_t port,
              bool serverType = false,
              std::string dstIp = "127.0.0.1",
              int timeoutMsec = 100);

    /// Read data.
```

```
    int read(uint8_t* data,
             int size,
             sockaddr_in* srcAddr = nullptr);

    /// Send data.
    int send(uint8_t* data, int size, sockaddr_in* dstAddr = nullptr);

    /// Check if UDP socket open.
    bool isOpen();

    /// Close UDP socket.
    void close();

    /// Get IP of data source.
    std::string getIp(sockaddr_in* srcAddr);

    /// Get UDP port of data source.
    int getPort(sockaddr_in* srcAddr);
};
```

getVersion method

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

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

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

```
std::cout << "UdpSocket version: " << cr::clib::UdpSocket::getVersion() << std::endl;
```

Console output:

```
UdpSocket version: 3.1.2
```

open method

The **open(...)** method initializes UDP socket. Method declaration:

```
bool open(uint16_t port, bool serverType = false, std::string dstIp = "127.0.0.1", int
timeoutMsec = 100);
```

Parameter	Value
port	UDP port. Must have values from 0 to 65535.
serverType	Socket type: TRUE - socket will be able to read and write data, FALSE - socket will be able only send data.
dstIp	Destination IP address to send data.

Parameter	Value
timeoutMsec	Wait data timeout. Method sets timeout to UDP socket properties. Timeout determines behavior of **read(...)** method: method will wait input data maximum **timeoutMsec** milliseconds and will return negative results if no input data.

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

read method

The **read(...)** method designed to read (wait) input data. After receiving input data the method will return control immediately or will return control after timeout (set in open(...) method) if no input data. Method declaration:

```
int read(uint8_t* data, int size, sockaddr_in* srcAddr = nullptr);
```

Parameter	Value
data	Pointer to data buffer.
size	Size of data buffer and maximum data size to read from socket.
srcAddr	Optional pointer to address structure. Method returns address structure of data source. User can use it to send data back to data source.

Returns: Number of bytes or **-1** if no input data or timeout expired (set in **open(...)** method) or UDP socket not open.

send method

The **send(...)** method sends data. Method declaration:

```
int send(uint8_t* data, int size, sockaddr_in* dstAddr = nullptr);
```

Parameter	Value
data	Pointer to data buffer.
size	Size of data to send.
dstAddr	Optional pointer to address structure. If address structure provide method will send data to this address.

Returns: Number of bytes sent or **-1** if data not sent or UDP socket not open.

isOpen method

The **isOpen()** method returns UDP socket open status. Method declaration:

```
bool isOpen();
```

Returns: TRUE if UPD socket open or FALSE if not.

close method

The **close()** method closes socket if it open. Method declaration:

```
void close();
```

getIp method

The **getIp(...)** method extracts IP address from address structure. Method declaration:

```
std::string getIp(sockaddr_in* srcAddr);
```

Parameter	Value
srcAddr	Pointer to address structure.

Returns: IP string.

getPort method

The **getPort(...)** method extracts UDP port from address structure. Method declaration:

```
int getPort(sockaddr_in* srcAddr);
```

Parameter	Value
srcAddr	Pointer to address structure.

Returns: UDP port.

Examples

Data sender

Test application shows how to create socket only to send data. Test application send random data periodically.

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

using namespace std;
using namespace cr::clib;
using namespace std::chrono;

int main(void)
{
    cout<< "Data sender v" << UdpSocket::getVersion() << endl << endl;

    string ip = "";
    cout << "Set destination IP: ";
    cin >> ip;

    int port = 0;
    cout << "Set UDP port: ";
    cin >> port;

    int cyclePeriodMsec = 0;
    cout << "Set sending data period msec: ";
    cin >> cyclePeriodMsec;

    int numBytes = 0;
    cout << "Set num bytes to send [0-8192]: ";
    cin >> numBytes;

    // Open UDP socket: client mode (only to send data).
    UdpSocket udpSocket;
    if (!udpSocket.open(port, false, ip))
        return -1;

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

        // Send data.
        cout << udpSocket.send(data, numBytes) << " bytes sent" << endl;

        // Wait according to parameters.
        int waitTime = (int)duration_cast<milliseconds>(system_clock::now() -
                                                        startTime).count();
```

```
        waitTime = cyclePeriodMsec - waitTime;
        if (waitTime > 0)
            this_thread::sleep_for(milliseconds(waitTime));
        startTime = system_clock::now();
    }
    return 1;
}
```

Data receiver

Test application shows how to create socket to read and send. Test application receives and shows info about sender.

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

using namespace std;
using namespace cr::clib;
using namespace std::chrono;

int main(void)
{
    cout<< "Data receiver v" << UdpSocket::getVersion() << endl << endl;

    int port = 0;
    cout << "Set UDP port: ";
    cin >> port;

    int timeoutMsec = 0;
    cout << "Set wait data timeout, msec: ";
    cin >> timeoutMsec;

    // Open UDP socket: server mode (to receive data).
    UdpSocket udpSocket;
    if (!udpSocket.open(port, true, "127.0.0.1", timeoutMsec))
        return -1;

    // Main loop.
    uint8_t data[8192];
    while (true)
    {
        // Read data. Max wait time = timeoutMsec.
        struct sockaddr_in addr;
        int bytes = udpSocket.read(data, 8192, &addr);

        // Check input data size.
        if (bytes <= 0)
        {
            cout << "No input data" << endl;
            continue;
```

```
        }

        // Show sender info.
        cout << bytes << " bytes read from " <<
        udpSocket.getIp(&addr) << "/" << udpSocket.getPort(&addr) << endl;
    }
}
```

Build and connect to your project

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

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

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

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

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

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

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(${PARENT}_SUBMODULE_UDP_SOCKET                      ON  CACHE BOOL "" FORCE)
if (${PARENT}_SUBMODULE_UDP_SOCKET)
    SET(${PARENT}_UDP_SOCKET                            ON  CACHE BOOL "" FORCE)
    SET(${PARENT}_UDP_SOCKET_EXAMPLES                   OFF CACHE BOOL "" FORCE)
endif()

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

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

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

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

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

Done!