# UdpSocket C++ library

**v3.1.0**

## Table of contents

## Overview

**UdpSocket** C++ library provides methos 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.

## Versions

**Table 1** - Library versions.

| Version | Release date | What's new |
|---|---|---|
| 1.0.0 | 10.10.2021 | First version |

| Version | Release date | What's new |
|---|---|---|
| 2.0.0 | 17.05.2022 | - Class interface changed.<br>- Added Tracker library to print debug info. |
| 3.0.0 | 13.05.2023 | - Class interface changed.<br>- Tracer library excluded. |
| 3.0.1 | 13.05.2023 | - Fixed send data to external address. |
| 3.1.0 | 27.08.2023 | - License added.<br>- Repository made public. |

# UdpSocket class description

## Class declaration

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

```
namespace cr
{
namespace clib
{
/**
 * @brief UDP Socket class.
 */
class UdpSocket
{
public:

    /**
     * @brief Get current library version.
     * @return String of current library version in format "X.Y.Z".
     */
    static std::string getVersion();

    /**
     * @brief Class constructor.
     */
    UdpSocket();

    /**
     * @brief Class destructor.
     */
    ~UdpSocket();

    /**
     * @brief Open UDP socket.
     * @param port UDP port.
     * @param serverType TRUE to send/receive data, FALSE only to send data.
     *        (socket will not be bind).
     * @param timeoutMsec Wait data timeout in milliseconds.
     * @return TRUE in socket open or FALSE if not.
     */
    bool open(uint16_t port,
              bool serverType = false,
              std::string dstIp = "127.0.0.1",
              int timeoutMsec = 100);

    /**
     * @brief Read data.
     * @param data Pointer to data buffer to copy data (not nullptr).
     * @param size Size of buffer.
     * @param srcAddr Pointer to structure from which the data was read.
     * @return Number of read bytes or return -1 in case error.
     */
    int read(uint8_t* data,
             int size,
             sockaddr_in* srcAddr = nullptr);

    /**
     * @brief Send data.
     * @param data Pointer to data to send.
     * @param size Size of data to send.
     * @param dstAddr Pointer to structure to data to send.
     * @return Number of bytes sent or return -1 if UDP socket not open.
     */
    int send(uint8_t* data, int size, sockaddr_in* dstAddr = nullptr);

    /**
     * @brief Check if UDP socket open.
     * @return TRUE if socket open or FALSE if not.
     */
    bool isOpen();

    /**
     * @brief Close UDP socket.
     */
    void close();

    /**
     * @brief Get IP of data source.
     * @param srcAddr Pointer to structure from which the data was read.
     * @return IP of data source.
     */
    std::string getIp(sockaddr_in* srcAddr);

    /**
     * @brief Get UDP port of data source.
     * @param srcAddr Pointer to structure from which the data was read.
     * @return UDP port of data source.
     */
    int getPort(sockaddr_in* srcAddr);
};
}
}
```

## getVersion method

**getVersion()** method return 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;
```

## open method

**open(...)** method designed to initialize 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. |
| 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

**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. |

| Parameter | Value |
|---|---|
| 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

**send(...)** method designed to send 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

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

```
bool isOpen();
```

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

## close method

close() method designed to close socket if it open. Method declaration:

```
void close();
```

## getIp method

**getIp(...)** method designed to extract 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

**getPort(...)** method designed to extract 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"

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

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

    // Enter destination IP.
    string ip = "";
    cout << "Enter destination IP: ";
    cin >> ip;

    // Enter UDP port.
    int port = 0;
    cout << "Enter UDP port: ";
    cin >> port;

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

    // Enter number of bytes.
    int numBytes = 0;
    cout << "Enter num bytes to send [0-8192]: ";
    cin >> numBytes;

    // Init UDP socket: clietn (only to send data), default destination IP.
    UdpSocket udpSocket;
    if (!udpSocket.open(port, false, ip))
    {
        cout << "ERROR: Can't init UDP socket. Exit." << endl;
        this_thread::sleep_for(seconds(1));
        return -1;
    }

    // Init variables.
    uint8_t* data = new uint8_t[numBytes];

    // Main loop.
    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 input data.

```
#include <iostream>
```

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

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

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

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

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

    // Init UDP socket: server, default destination IP.
    UdpSocket udpSocket;
    if (!udpSocket.open(port, true, "127.0.0.1", timeoutMsec))
    {
        cout << "ERROR: Can't init UDP socket. Exit." << endl;
        this_thread::sleep_for(seconds(1));
        return -1;
    }

    // Init variables.
    const int bufferSize = 1024;
    uint8_t data[bufferSize];

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

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

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