UdpDataChannel C++ library

v1.1.0

Table of contents

Overview

UdpDataChannel C++ library provide reliable point-to-point communication between two applications based on UDP. The library includes two primary classes: **UdpDataClient** (connects to server) and **UdpDataServer** (wait connection from client). Both classes provide two-way communication. The library uses C++17 standard and only depends on open source UdpSocket (source code included, Apache 2.0 license) which provide functions to work with UDP sockets. This library compatible with Linux and Window. **Server principles:** the server initializes the user-specified UDP port and waits for the client to connect from any port. The server supports connection of only one client. When a client connects the server remembers the client's IP and UDP port to exchange messages with them. The client sends special commands to connect. **Client principles:** the client initialized any first available UDP port in the OS. After initialization client sends connection messages to server port (to connect the user must say to client the server's UDP port and IP). Once connection established the client can exchanges messages with server. To send data server and client split data into separate UDP packets with special header data. Receiver (client or server) collect UDP packets and detect when whole input data being send by sender collected. After receiver sends confirmation. Sender sends data twice until it will get confirmation from receiver. The library can handle intensive data exchange close to channel bandwidth limit. The library allows user to set maximum channel bandwidth to prevent network overloading. Sender (client or server) controls interval between UDP packets being sent.

Versions

Table 1 - Library versions.

Version	Release date	What's new
1.0.0	20.11.2023	- First version of the library.
1.1.0	12.02.2024	- Repository structure reorganized. - Added separate classes for client and server. - Documentation updated. - Example added.

Library files

The library supplied by source code only. The user would be given a set of files in the form of a CMake project (repository). The repository structure is shown below:

```
CMakeLists.txt ------------------- Main CMake file of the library.
3rdparty ------------------------- Folder with third-party libraries.
    CMakeLists.txt --------------- CMake file, which includes third-party libraries.
    UdpSocket -------------------- Source code of the UdpSocket library.
client --------------------------- Folder with the UdpDataClient class.
    CMakeLists.txt --------------- CMake file for client.
    UdpDataClient.cpp ------------ Source code file of UdpDataClient class.
    UdpDataClient.h -------------- Header file includes UdpDataClient class declaration.
```

```
    UdpDataClientVersion.h ------- Header file which includes version of the client.
    UdpDataClientVersion.h.in ---- CMake service file to generate version file.
clienttest ----------------------- Folder for client test application.
    CMakeLists.txt --------------- CMake file for client test application.
    main.cpp --------------------- Source code file of client test application.
server --------------------------- Folder with the UdpDataServer class.
    CMakeLists.txt --------------- CMake file for server source files.
    UdpDataServer.cpp ------------ Source code file of UdpDataServer class.
    UdpDataServer.h -------------- Header file includes UdpDataServer class declaration.
    UdpDataServerVersion.h ------- Header file which includes version of the server.
    UdpDataServerVersion.h.in ---- CMake service file to generate version file.
servertest ----------------------- Folder for server test application.
    CMakeLists.txt --------------- CMake file for server application.
    main.cpp --------------------- Source code file of server test application.
example -------------------------- Folder for example application.
    CMakeLists.txt --------------- CMake file for example application.
    main.cpp --------------------- Source code file of example application.
```

UdpDataClient class description

Class declaration

UdpDataClient class declared in **client/UdpDataClient.h** file. Class declaration:

```
class UdpDataClient
{
public:

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

    /// Class constructor.
    UdpDataClient();

    /// Class destructor.
    ~UdpDataClient();

    /// Copy constructor for a client.
    UdpDataClient(const UdpDataClient& src);

    /// Initialize client object specifying the server's address and port.
    bool init(std::string serverIp, uint16_t serverPort,
              int channelBandwidthKbps = 1000000);

    /// Close client.
    void close();

    /// Get input data from connected client.
    bool get(uint8_t* data, int bufferSize, int& dataSize, int timeoutMsec = 0);

    /// Send data to the client.
```

```
    bool send(uint8_t* data, int size);

    /// Get connection status.
    bool isConnected();

    /// Get channel status.
    bool isInit();
};
```

getVersion method

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

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

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

```
std::cout << "UdpDataClient class version: " << cr::clib::UdpDataClient::getVersion();
```

Console output:

```
UdpDataClient class version: 1.1.0
```

init method

The **init(...)** method initializes client with server IP, UDP port and optional channel bandwidth. The client choses first available in OS UDP port from 20000 to 65535 range. After UDP port initialization the client starts communication threads and will start connection to server. Method declaration:

```
bool init(std::string serverIp, uint16_t serverPort, int channelBandwidthKbps = 1000000);
```

Parameter	Description
serverIp	Server IP.
serverPort	Server UDP port.
channelBandwidthKbps	Channel bandwidth in kilobits per second. The client will control time interval between UDP packets to prevent network overload.

Returns: TRUE if the client was successfully initialized or FALSE if not.

close method

The **close(...)** method closes client and releases all resources. This method stops all threads, closes all sockets and also releases all memory allocated in data buffers. Method declaration:

```
void close();
```

get method

The **get(...)** method returns received data. The client has internal buffer (size of 16 elements). The client put received data to the buffer. When user call get(...) method it returns data from buffer. The buffer can hold several portions of data (up to 16). In this case, the method will return the earliest one received from the server. This way the data will not be lost if the user does not have time to read it. If no input data from server the method will wait until it will come or particular timeout. The method is thread-safe and can be called from multiple threads. Method declaration:

```
bool get(uint8_t* data, int bufferSize, int& dataSize, int timeoutMsec = 0);
```

Parameter	Description
data	A pointer to a buffer where the received data will be stored.
bufferSize	The size of the buffer allocated for storing the received data. If input data has bigger size than buffer the method will return FALSE.
dataSize	Size of input data. If no input data size will be 0.
timeoutMsec	The timeout period for receiving data, in milliseconds. Values: **<0** - wait until data will come. **0** - just check if we have new data. **>0** - wait timeout, msec.

Returns: TRUE if there is new data or FALSE if not.

send method

The **send(...)** method sends data to server. The client has internal buffer (size of 16 elements). The client put data for server to the buffer. Internal thread reads data from buffer, splits by UDP packets and sends to server. This way the data will not be lost if the user calls the method multiple times. The method is thread-safe and can be called from multiple threads. Method declaration:

```
bool send(uint8_t* data, int size);
```

Parameter	Description
data	A pointer to the data buffer containing the data to be sent.

Parameter	Description
size	Data size to send. Maximum size is 67108863 bytes. If size more than 67108863 bytes the method will return FALSE.

Returns: TRUE if the data accepted to send or FALSE if not.

isConnected method

The **isConnected(...)** method return connection status. After initialization the client tries connect to server. Once the client connected to server the method will be returning TRUE. Method declaration:

```
bool isConnected();
```

Returns: TRUE if the client is currently connected to the server or FALSE if not.

isInit method

The **isInit(...)** method return client initialization status. Method declaration:

```
bool isInit();
```

Returns: TRUE if the client is initialized or FALSE if not.

UdpDataServer class description

Class declaration

UdpDataServer interface class declared in **client/UdpDataServer.h** file. Class declaration:

```
class UdpDataServer
{
public:

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

    /// Class constructor.
    UdpDataServer();

    /// Class destructor.
    ~UdpDataServer();

    /// Copy constructor for a server.
    UdpDataServer(const UdpDataServer& src);
```

```
    /// Initialize server object specifying the port number and bandwidth.
    bool init(uint16_t port, int channelBandwidthKbps = 1000000);

    /// Close server. Complete all open threads and clear data buffers.
    void close();

    /// Get input data from connected client.
    bool get(uint8_t* data, int bufferSize, int& dataSize, int timeoutMsec = 0);

    /// Send data.
    bool send(uint8_t* data, int size);

    /// Get connection status.
    bool isConnected();

    /// Get channel status.
    bool isInit();
};
```

getVersion method

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

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

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

```
std::cout << "UdpDataServer class version: " << cr::clib::UdpDataServer::getVersion();
```

Console output:

```
UdpDataServer class version: 1.1.0
```

init method

The **init(...)** method initializes server with UDP port and optional channel bandwidth. After UDP port initialization the server starts communication threads and will start listening connection requests from client. Method declaration:

```
bool init(uint16_t port, int channelBandwidthKbps = 1000000);
```

Parameter	Description
port	Server UDP port.
channelBandwidthKbps	Channel bandwidth in kilobits per second. The server will control time interval between UDP packets to prevent network overload.

Returns: TRUE if the server was successfully initialized or FALSE if not.

close method

The **close(...)** method closes server and releases all resources. This method stops all threads, closes all sockets and also releases all memory allocated in data buffers. Method declaration:

```
void close();
```

get method

The **get(...)** method returns received data. The server has internal buffer (size of 16 elements). The server put received data to the buffer. When user call get(...) method it returns data from buffer. The buffer can hold several portions of data (up to 16). In this case, the method will return the earliest one received from the client. This way the data will not be lost if the user does not have time to read it. If no input data from client the method will wait until it will come or particular timeout. The method is thread-safe and can be called from multiple threads. Method declaration:

```
bool get(uint8_t* data, int bufferSize, int& dataSize, int timeoutMsec = 0);
```

Parameter	Description
data	A pointer to a buffer where the received data will be stored.
bufferSize	The size of the buffer allocated for storing the received data. If input data has bigger size than buffer the method will return FALSE.
dataSize	Size of input data. If no input data size will be 0.
timeoutMsec	The timeout period for receiving data, in milliseconds. Values: **<0** - wait until data will come. **0** - just check if we have new data. **>0** - wait timeout, msec.

Returns: TRUE if there is new data or FALSE if not.

send method

The **send(...)** method sends data to client. The server has internal buffer (size of 16 elements). The server put data for client to the buffer. Internal thread reads data from buffer, splits by UDP packets and sends to client. This way the data will not be lost if the user calls the method multiple times. The method is thread-safe and can be called from multiple threads. Method declaration:

```
bool send(uint8_t* data, int size);
```

Parameter	Description
data	A pointer to the data buffer containing the data to be sent.
size	Data size to send. Maximum size is 67108863 bytes. If size more than 67108863 bytes the method will return FALSE.

Returns: TRUE if the data accepted to send or FALSE if not.

isConnected method

The **isConnected(...)** method return connection status. After initialization the server waits for client connection. Once the client connected the method will be returning TRUE. Method declaration:

```
bool isConnected();
```

Returns: TRUE if the client is currently client connected or FALSE if not.

isInit method

The **isInit(...)** method return server initialization status. Method declaration:

```
bool isInit();
```

Returns: TRUE if the client is initialized or FALSE if not.

Build and connect to your project

Typical commands to build the library:

```
cd UdpDataChannel
git submodule update --init --recursive
mkdir build
cd build
cmake ..
make
```

If you want to connect **UdpDataChannel** to your CMake project as source code you can follow these steps. For example, if your repository has structure:

```
CMakeLists.txt
src
    CMakeList.txt
    yourLib.h
    yourLib.cpp
```

Create folder **3rdparty** in you repository and copy UdpDataChannel repository folder there. The new structure of your repository:

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

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(${PARENT}_SUBMODULE_UDP_DATA_CHANNEL                ON  CACHE BOOL "" FORCE)
if (${PARENT}_SUBMODULE_UDP_DATA_CHANNEL)
    SET(${PARENT}_UDP_DATA_CLIENT                       ON  CACHE BOOL "" FORCE)
    SET(${PARENT}_UDP_DATA_SERVER                       ON  CACHE BOOL "" FORCE)
    SET(${PARENT}_UDP_DATA_CLIENT_TEST                  OFF CACHE BOOL "" FORCE)
    SET(${PARENT}_UDP_DATA_SERVER_TEST                  OFF CACHE BOOL "" FORCE)
    SET(${PARENT}_UDP_DATA_CHANNEL_EXAMPLE              OFF CACHE BOOL "" FORCE)
endif()

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

File **3rdparty/CMakeLists.txt** adds folder **UdpDataChannel** to your project and excludes test applications from compiling. The new structure of your repository:

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

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

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

Done!

Example

This examples demonstrates how to use server and client. The application creates thread for server which waits data from client and send responses. Client sends random data to server in separate thread. Source code of example:

```
#include <iostream>
#include <thread>
#include <cstdint>
#include "UdpDataServer.h"
#include "UdpDataClient.h"

void serverThreadFunction()
{
    // Init server.
    cr::clib::UdpDataServer server;
    if (!server.init(57000))
        return;

    // Thread loop.
    uint8_t buffer[256];
    while (true)
    {
        // Wait data from client for 2 sec.
        int size = 0;
        if (!server.get(buffer, 256, size, 2000))
            std::cout << "No input data from client" << std::endl;

        // Send data back to client.
        if (!server.send(buffer, size))
            std::cout << "Can't send data to client" << std::endl;
```

```
    }
}

int main(void)
{
    // Run server thread.
    std::thread serverThread(&serverThreadFunction);

    // Init client.
    cr::clib::UdpDataClient client;
    if (!client.init("127.0.0.1", 57000))
        return -1;

    // Main loop.
    uint8_t buffer[256];
    while (true)
    {
        // Wait data from server.
        int size = 0;
        if (!client.get(buffer, 256, size, 2000))
            std::cout << "No input data from server" << std::endl;
        else
            std::cout << size << " bytes from server" << std::endl;

        // Prepare random data for server.
        size = (rand() % 128) + 1;
        memset(buffer, (rand() % 255), size);

        // Send data to server.
        if (!client.send(buffer, size))
            std::cout << "Can't send data to server" << std::endl;
    }

    return 1;
}
```