KØBENHAVNS ERHVERVSAKADEMI
Copenhagen School of Design and Technology

Paul Dean Samsig
4th semester int. net a
Final Project; Glenn Muhlack
2015-01-02

ChatTwo – Building a chat program

KØBENHAVNS ERHVERVSAKADEMI
Copenhagen School of Design and Technology

Paul Dean Samsig
4th semester int. net a
Final Project; Glenn Muhlack
2015-01-02

ChatTwo – Building a chat program

Solemn Declaration

I solemnly declare that I have personally and independently created this report. I have clearly marked any and all quotes in the text as such, and neither the report nor any essential parts of it are at present or have previously been submitted for any other examination.

I am aware that any violation of the rules on academic integrity shall be treated in accordance with Article 19 of the Danish Order No 1016 of 24 August 2010 on Tests and Examinations in vocational educations.

[Student's signature]

_____

[Student's name, date]

Table of Contents

Introduction

For years I have been using chat programs to communicate with my family, friends, and coworkers. Mainstream ones such as AOL Instant Messenger, MSN Messenger, Yahoo! Messenger, Skype, and Google Talk; and now even Facebook's Instant Messenger. However, these programs all had problems that would interfere with their intended function. (Can always list some of the issues you have with each) I would think about making my own, that could have the features I need, but without the problems.

After my Third semester programming project, where I was able to create a simple chatroom program; which motivated me to try and make full chat program.

Project Scope

In this report, I want to illustrate how I believe a chat program should work; and how I have been working towards making my vision a reality.

The name I have chosen to christen my program is ChatTwo. This is my way of naming things, my online handle is Deantwo, so when I name something it is always something like ThingTwo.

For the foreseeable future, I will not be delving into heavy security for the program. Security, like password encryption and encrypting network traffic, can always be added later once I have the core program up and running. However, I will be adding in a basic message authentication code (MAC) and simple hashing of passwords, just to get started.

Problem formulation

Can I make something as good as the mainstream chat programs in the time that I have? I'd like to say yes, but let me spoil it a little and say that it at least isn't done yet.

If I want to make the client to client communication peer-to-peer, I need to figure out how to get through a NAT router?

I don't want to focus on security this early on, so what is the minimal amount of security needed?

Will it cost a lot of money to run the server needed to offer this service to the world? I don't want to make something that will end up costing millions to maintain.

Theory

Communication

First thing I needed to figure out is how to communicate between the clients. The standard way would be to use a server as a relay; but this would not be ideal in the sense of scalability.

If we say that I get more and more users as time goes, I'll get more and more traffic to and from the server. This means that if I want to keep delivering a good service I need a better and better bandwidth to the server to meet the increasing traffic.

Peer-to-peer

To combat this I will attempt to make a peer-to-peer communication system. The only thing the server will do in this case is handle online statuses and forward the IP addresses of the available clients. This means that clients will communicate directly with each other once they have received their addresses from the server.

Pros:

- Traffic to the server will be cut down a lot.
- The chat between clients can't be tracked by a rogue server.

Cons:

- Your IP address is, in theory, available to the person you are talking with and vice versa. This could lead to some privacy issues.
- Wire-less devices' (laptops, smartphones, etc.) IP address may change when roaming, possibly causing some connection issues.

Network Traffic

I have the most experience working with UDP traffic, designing some kind of transport layer on my own will be a fun challenge. UDP will also be easier when it comes to NAT hole punching.

NAT Hole Punching

Hole punching a is NAT traversal technic. It relays on the NAT's behavior of setting up temporary port forwarding with its NAT table when it handles an outgoing packet.

So if you have sent out an UDP packet using the source port of 900. The router will forward all incoming UDP packets with the destination port of 900 to your computer for a short while.

A problem then arises when the internal and external ports aren't the same, for example when the router already has another service mapped to port 900. This can be worked around by using a remote server that you can connect to and have it report back with your external port number.

This technic works somewhat similar for a TCP communication. But there is more to worry about in the term of connection limits. I am hoping to implement a way to use TCP at some point in the future.

User Accounts

Much like the mainstream chat programs, I will attempt to make a user system. This mean each user will have a unique username that others can't take. Each user will choose a password with their username that will make them the only ones that can access their username.

Security

The only security that will be implemented in the beginning is a message authentication code (MAC) system. This is required to make sure that another user can't hijack your connection with the server.

Message Authentication Code

A message authentication code (MAC) works by having both ends of a conversation have a code word. When they send a message, they add this code word in so that the receiver knows that the message is indeed from the correct person.

Of course in the computer work this is a little more complicated than that because else hijackers would just copy the code word from the any message that he may see.

To make this work the code word has to be uniquely tied to the message, for example if you took the first letter of each word in the message and used that as a key word. Now the code word depends on what is in the message, but is easy to figure out.

To fix this issue we mix the two ideas together and put both code words in. But to make sure that no one can reverse the code words and figure them out, we crush them in what is known as a hashing function. When something is hashed you can never get the original content back. Much like getting your car crushed into a little cube, no one will be able to say what the car looked like, and two diffident cars won't look the same after being crushed.

Only difference between a crushed car and hashed data is that we can create the same hash again if we have the same data. So we take the message that we were going to send, take a copy of it and crush it in our hashing function, getting a unique code. We combine this code with our code word and crush them together in the hash function again; we are not left with a MAC. We add the MAC to the end of the original message and send it.

In the other end we now have the message and the MAC, to ensure that the MAC is correct we take the message we received and crush it, then add the code word that we already have and crush it again. Now we have our own MAC, and if our MAC is identical to the MAC we received with the message, we know that it came from the correct person and that the message was not changed in anyway way during transit.

Construction

Hardware

The only hardware needed for my project is a server that is accessible from the internet. This can easily be done with my desktop computer at home. I can set my home router to port forward the desired port to my desktop computer easily.

Can then use my laptop as a client from anywhere by connecting to my home router's external IP address.

Software

Programming Language

C# is the only programming language I have really had in school and feel comfortable with.

I have also made multiple smaller programs for school that has code that I can reuse.

SQL Database

During programming classes in KEA we had some assignments with MySQL, so already had XAMPP installed, XAMPP is an easy to install webserver that has PHP and MySQL included. I ended up installing XAMPP on my desktop rather than a standard MySQL server because the official MySQL installer for Windows ended up throwing exceptions at me when I tried to install it.

Reversion Control

In my free time, I taught myself the basics of GitHub and have found it very useful for developing. I decided to use it for my project as well. This also means that my sources-code is available to the public and other users can report errors or make suggestions easily, at least if anyone knew about my project.

I didn't use it as much as I could have, since I had some confusion on how to solve some of my problem and my late uploading the projects to the site.

Prototype

Shared Classes

I wanted to be able to have some code that was shared between both the client and the server application. The shared classes I used on both applications are: ByteHelper, IpCommunication, and ChatTwo_Protocol.

ByteHelper is a class I created for an application I made last year. I mostly just needed it for the SubArray and ConcatinateArray methods that it has. I then added my hashing methods to it as it seemed like the best place for it.

IpCommunication is my attempt at making layered programming. The other classes simple forward the data they need to send to this class. It then does all the TCP/IP work and tries to ensure that the message is delivered, much like the Transport layer of the TCP/IP model.

ChatTwo_Protocol was first of all my first attempt at making client-server communication. I then later split the class into three classes: ChatTwo_Protocol, ChatTwo_Client_Protocol, and ChatTwo_Server_Protocol. ChatTwo_Protocol contain all the shared methods that are used by both client and server.

Plug n' Play Class Design

At the beginning of my project, I decided that I wanted to have the classes not be strictly bond to each other. To do this I made my first ever try at making custom events.

This mean that the classes I created don't make any calls to the main form, making it fairly easy to simply add the classes to a new project in the future without having to rewrite the classes.

The best example of this is the IpCommunication class. It has an input and an output, and both of these are events. Here is the input:

```
// Fire an OnPacketReceived event.
PacketReceivedEventArgs args = new PacketReceivedEventArgs();
args.Sender = remoteSender;
args.Data = receivedBytes;
OnPacketReceived(args);
```

And here is the output:

```
public void SendPacket(object sender, PacketTransmissionEventArgs args)
{
    ControlledPacket ctrlPacket = new ControlledPacket();
    ctrlPacket.Recipient = args.Destination;
    ctrlPacket.Data = args.PacketContent;

    _messageSendingControlList.Add(ctrlPacket);
}
```

Thanks to this I am able to use the IpCommunication class in any application with minimal work. All I need is these four lines and I can make it work in both the client and server applications:

```
UdpCommunication _client = new UdpCommunication();
_client.PacketReceived += ChatTwo_Client_Protocol.MessageReceivedHandler;
ChatTwo_Client_Protocol.MessageTransmission += _client.SendPacket;
_client.Start(0);
```

IpCommunication

IpCommunication is my transport layer style class, it basically just the link between my protocol and the internet. Rather than have my protocol class also be the sender of the data, this class alone tries to ensure that the UDP packets are delivered successfully.

Transmission Control

```
protected void PacketTransmissionControl() // Threaded looping method.
{
    try
    {
        while (_online)
        {
            CheckPacketControlList();
            Thread.Sleep(200);
        }
    }
    catch (Exception ex)
    {
        System.Diagnostics.Debug.WriteLine("### " + _threadPacketSending.Name + " has crashed:");
        System.Diagnostics.Debug.WriteLine("### " + ex.Message);
        System.Diagnostics.Debug.WriteLine("### " + ex.ToString());
    }
}

protected void CheckPacketControlList()
{
    if (_messageSendingControlList.Count != 0)
    {
        List<ControlledPacket> temp = _messageSendingControlList.FindAll(x => (x.LastTry == null
            || (DateTime.Now - x.LastTry).TotalMilliseconds > 400) && x.Attempts < 5);
        foreach (ControlledPacket ctrlPacket in temp)
        {
            if (SendControlledPacket(ctrlPacket))
            {
                ctrlPacket.LastTry = DateTime.Now;
                ctrlPacket.Attempts++;
                if (ctrlPacket.Attempts == 5)
                    _messageSendingControlList.Remove(ctrlPacket);
            }
        }
    }
}
```

The class takes an input in the form of a byte array and an IPEndPoint, adds them both to a list and tries to send it. If the packet has not been ACKed by the other end, the class sends the packet again. In my prototype it only tries to send the packet 5 times before silently marking it as failed, still thinking about how to improve this behavior.

When IpCommunication receives an incoming packet, it first takes a hash of the whole byte content and send it back to the sender as an ACK, indication that the packet was received. It then checks a list of the last 5 packet that has been received, to make sure it is not a duplicate.

If it's not a duplicate if fire the PacketReceived event. In the event's EventArgs, the byte content and sender's IPEndPoint of the packet is available to the receiving method.

I have plans to make IpCommincation also support TCP, but haven't had haven't used TCP before and afraid it would go against some of my ideas for NAT hole punching.

ACK

```
protected byte[] CreateAck(string hash)
{
    byte[] ackTag = new byte[] { 0xCE }; // 0xCE = 206
    byte[] ackBytes = ByteHelper.ConcatinateArray(ackTag, Convert.FromBase64String(hash), ackTag);
    return ackBytes;
}

protected string OpenAck(byte[] bytes)
{
    string ackHash = Convert.ToBase64String(bytes, 1, ByteHelper.HashByteLength);
    return ackHash;
}
```

The ACK packet is 22 bytes long and contains the hash of the full content of the received packet that it is ACKing. In both ends of the hash a tag byte is added, it is used by IpCommunication to correctly identify the packet as an ACK.

When an ACK is received, a search is done in the list of out-going packets, if a match is found; it is removed so it is not resend again. This mean that if

Database

The database was one of the first things I worked on for this project. Since I hadn't used SQL for anything like this before, I wanted to test it a little. Not sure it was a good idea to store the online status and socket information about users on the SQL server was a good idea, but it works for now.

Tables

```
CREATE TABLE `Users` (
    `ID` INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
    `Name` VARCHAR(30) NOT NULL UNIQUE,
    `Password` VARCHAR(28) NOT NULL,
    `Online` TINYINT(1) NOT NULL DEFAULT 0,
    `Socket` VARCHAR(51) NULL,
    `LastOnline` TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    `Registered` TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP
    );
```

The Users table is fair simple. Added some fun to have columns to it like `Registered` and `LastOnline`. I then later ended up using `LastOnline` entry to detect user timeouts, it work but feels wrong on some level, more about that below.

```
CREATE TABLE `Contacts` (
    `ID_1` INT NOT NULL,
    `ID_2` INT NOT NULL,
    `1To2` TINYINT(1) NOT NULL DEFAULT 0,
    `2To1` TINYINT(1) NOT NULL DEFAULT 0
    );
```

The Contacts table got a little complex. Rather than create a table for each user as I original had planned, I came up this this little idea. The table simply contains all relationships; a problem may arise if all users have everyone as a contact, because it would mean the contacts table would contain $\left((userCount-1)userCount/2\right)$ entries.

`ID_1` and `ID_2` are the IDs of the users, it does not matter which is which, and there should only ever be one pairing between two users. `1To2` and `2To1` are boolean values that indicate in which directions the relationship is.

```
CREATE TABLE `ServerStatus` (
    `Version` INT NOT NULL,
    `CreationDate` TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    `LastUpdated` TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP
    );
INSERT INTO `ServerStatus` (`Version`) VALUES(0);
```

The ServerStatus table is a simple table that will most likely only ever contain one row. This is simply to know when the server was created and what version it is, this is mostly for future backward compatibility.

Stored Procedures

I made some stored procedures for doing some of the more complicated queries. For example contacts table was a little complex so all queries to it will be using stored procedures.

```
CREATE DEFINER=CURRENT_USER PROCEDURE `StatusIntervalUpdate`()
    MODIFIES SQL DATA
BEGIN
    SELECT `ID` FROM `Users`
        WHERE (`Online` = 1)
          AND NOT (`LastOnline` BETWEEN timestamp(DATE_SUB(NOW(), INTERVAL 10 SECOND)) AND NOW());
    UPDATE `Users`
        SET `Online` = 0,
            `Socket` = NULL
        WHERE (`Online` = 1)
          AND NOT (`LastOnline` BETWEEN timestamp(DATE_SUB(NOW(), INTERVAL 10 SECOND)) AND NOW());
END $$
```

This is the attempt at using the User table's `LastOnline` column to detect timeouts. The procedure is executed every second and returns a list of all the users that have a `LastOnline` timestamp that is older than 10 seconds before changing their status to offline.

```
CREATE DEFINER=CURRENT_USER PROCEDURE `ContactsAdd`(
    IN `p_ID` INT,
    IN `p_ContactID` INT
)
    MODIFIES SQL DATA
BEGIN
    IF (SELECT EXISTS(SELECT 1 FROM `Contacts`
        WHERE (`ID_1` = p_ID AND `ID_2` = p_ContactID)
           OR (`ID_2` = p_ID AND `ID_1` = p_ContactID) LIMIT 1) as contactFound) = 1
    THEN
        UPDATE `contacts`
        SET `2To1` = IF(`ID_2` = p_ID, 1, `2To1`), `1To2` = IF(`ID_1` = p_ID, 1, `1To2`)
        WHERE (`ID_1` = p_ID AND `ID_2` = p_ContactID)
           OR (`ID_2` = p_ID AND `ID_1` = p_ContactID);
    ELSE
        INSERT INTO `contacts`(`ID_1`, `ID_2`, `1To2`, `2To1`)
        VALUES (p_ID, p_ContactID, 1, 0);
    END IF;
END $$
```

This procedure first check if a pairing for the two users already exist, and if it does simply

update it to have the requesting user be in a relationship towards the contact. If the pairing does not exist, it is created.

```
CREATE DEFINER=CURRENT_USER PROCEDURE `ContactsRemove`(
    IN `p_ID` INT,
    IN `p_ContactID` INT
)
    MODIFIES SQL DATA
BEGIN
    UPDATE `contacts`
    SET `2To1` = IF(`ID_2` = p_ID, 0, `2To1`), `1To2` = IF(`ID_1` = p_ID, 0, `1To2`)
    WHERE (`ID_1` = p_ID AND `ID_2` = p_ContactID)
       OR (`ID_2` = p_ID AND `ID_1` = p_ContactID);
    DELETE FROM `contacts`
    WHERE `1To2` = 0 AND `2To1` = 0;
END $$
```

This procedure searches for a pairing between the user and the contact, and then removes the direction from user to contact. As a little cleanup it then attempts to delete all entries that have no relation in either direction.

```
CREATE DEFINER=CURRENT_USER PROCEDURE `ContactsAll`(
    IN `p_ID` INT
)
    READS SQL DATA
BEGIN
    SELECT IF((`ID_1` = p_ID), `ID_2`, `ID_1`) AS ContactID,
           IF((`ID_1` = p_ID AND `1To2` = 1) OR (`ID_2` = p_ID AND `2To1` = 1), 1, 0) AS FromMe,
           IF((`ID_1` = p_ID AND `2To1` = 1) OR (`ID_2` = p_ID AND `1To2` = 1), 1, 0) AS ToMe
        FROM `Contacts`
        WHERE `ID_1` = p_ID OR `ID_2` = p_ID;
END $$
```

This procedure returns a row for each contact the user has, as well as two boolean values explaining if it is a mutual friendship or if one is waiting on the other to accept the request.

```
CREATE DEFINER=CURRENT_USER PROCEDURE `ContactsMutual`(
    IN p_ID INT
)
    READS SQL DATA
BEGIN
    SELECT IF((`ID_1` = p_ID), `ID_2`, `ID_1`) AS `ContactID`
        FROM `Contacts`
        WHERE (`ID_1` = p_ID OR `ID_2` = p_ID)
          AND `1To2` = 1 AND `2To1` = 1;
END $$
```

This procedure returns a list of all the user's contacts that have a mutual friendship. I would like to make a change to this procedure though, make it only return mutual contacts that are currently online.

DatabaseCommunication

DatabaseCommunication is, as the name suggest, the class I use to communicate with the database. I made the whole class static because I won't ever need multiple instances of it.

TestConnection

```
public static ConnectionTestResult TestConnection(string user, string password, string ip, int port)
{
    // Shorter timeout will make the user not have to wait as long.
    // (Does not seem to have much of an effect on the connection timeout.)
    const int timeout = 5;

    MySqlConnectionStringBuilder connBuilder = new MySqlConnectionStringBuilder();
    connBuilder.Add("User id", user);
    connBuilder.Add("Password", password);
    connBuilder.Add("Network Address", ip);
    connBuilder.Add("Port", port);
    //connBuilder.Add("Database", "ChatTwo");
    connBuilder.Add("Connection timeout", timeout);

    // Test1: Test connection to the server using the IP address from the settings. Add "Connection Timeout" (even though it seem not to work).
    using (MySqlConnection testConn = new MySqlConnection(connBuilder.ConnectionString))
    {
        // Test2: Test access to the database.
        MySqlCommand test2 = new MySqlCommand("USE `ChatTwo`;", testConn);
        test2.CommandTimeout = timeout;

        // Test3: Test access to the `Contacts` table and the `Users` table..
        MySqlCommand test3 = new MySqlCommand("SELECT * FROM `ChatTwo`.`Contacts` WHERE 0 = 1;SELECT * FROM `ChatTwo`.`Users` WHERE 0 = 1;", testConn);
        test3.CommandTimeout = timeout;

        // Test4: Test access to the `ServerStatus` table and get the version number.
        MySqlCommand test4 = new MySqlCommand("SELECT `Version` FROM `ChatTwo`.`ServerStatus`;", testConn);
        test4.CommandTimeout = timeout;
        int version = -1;

        // Run all tests.
        try
        {
            testConn.Open();
            test2.ExecuteNonQuery();
            test3.ExecuteNonQuery();
            MySqlDataReader reader = test4.ExecuteReader();
            if (reader.Read())
            {
                version = (int)reader["Version"];
            }
        }
        catch (MySqlException ex)
```

The TestConnection method is used to test if a connection would actually work before starting a real connection. It takes the needed parameters to create a connection string using the MySqlConnectionStringBuilder system and then sets up four tests:

1. The first test is to actually try and open the connection to the SQL server.
2. Then it will try to USE the database. This is to check if the database exists.
3. The third test to test if the `Users` and `Contacts` tables exist.
4. Last test is to check the version number in the `ServerStatus` table.

These four tests are then executed in a try case and a catch case catches any MySqlException that may be thrown if one of the tests fails.

```
            catch (MySqlException ex)
            {
                // If one of the tests fail, return an error message.
                switch (ex.Number)
                { // http://dev.mysql.com/doc/refman/5.6/en/error-messages-server.html
                    case 0:
                        if (ex.Message.Contains("Access denied"))
                        {
                            // Login failed
                            return ConnectionTestResult.FailLogin;
                        }
                        else
                        {
                            // SQL query timed out
                            return ConnectionTestResult.NoConnection;
                        }
                    case 1042:
                        // (ER_BAD_HOST_ERROR) Message: Can't get hostname for your address
                        return ConnectionTestResult.NoConnection;
                    case 1044:
                        // (ER_DBACCESS_DENIED_ERROR) Message: Access denied for user '%s'@'%s' to database '%s'
                        return ConnectionTestResult.NoPermission;
                    case 1049:
                        // (ER_BAD_DB_ERROR) Message: Unknown database '%s'
                        return ConnectionTestResult.MissingDatabase;
                    case 1146:
                        // (ER_NO_SUCH_TABLE) Message: Table '%s.%s' doesn't exist
                        return ConnectionTestResult.MissingTable;
                    default:
                        // Unknown SQL error
                        return ConnectionTestResult.UnknownError;
                }
            }
            finally
            {
                if (testConn.State != ConnectionState.Closed)
                    testConn.Close();
            }

            // If the version is old, suggest an update.
            if (version < _version)
                return ConnectionTestResult.OutDated;
        }

        // If nothing bad happens, tell the user the program is ready.
        return ConnectionTestResult.Successful;
    }
```

When a MySqlException is thrown the catch will run. There I have a simple switch case that uses the MySqlException's error code number to identify what kind of error it is. I have then through some testing found the possible MySqlExceptions that may be thrown and keyed them each to an enum called ConnectionTestResult.

Only one of possible errors is non-exception related, and that is the ConnectionTestResult.OutOfDate. It is used for future backward compatibility.

CreateDatabase

```
public static bool CreateDatabase(string user, string password, string ip, int port)
{
    int cmdResult = 0;

    MySqlConnectionStringBuilder connBuilder = new MySqlConnectionStringBuilder();
    connBuilder.Add("User id", user);
    connBuilder.Add("Password", password);
    connBuilder.Add("Network Address", ip);
    connBuilder.Add("Port", port);
    //connBuilder.Add("Database", "ChatTwo");
    //connBuilder.Add("Connection timeout", 5);

    using (MySqlConnection tempConn = new MySqlConnection(connBuilder.ConnectionString))
    {
        using (MySqlCommand cmd = new MySqlCommand(
            "CREATE DATABASE IF NOT EXISTS `ChatTwo`;" + Environment.NewLine +
            "USE `ChatTwo`;" + Environment.NewLine +

            (See the full code.)

            "END;"
            , tempConn))
        {
            try
            {
                tempConn.Open();
                // Execute SQL command.
                cmdResult = cmd.ExecuteNonQuery();
            }
            finally
            {
                if (tempConn.State != System.Data.ConnectionState.Closed)
                    tempConn.Close();
            }
        }
    }
    return (cmdResult != 0);
}
```

CreateDatabase is made to make it easier for users that don't know how to work a MySQL database. It creates the whole database from scratch, all that is needed is a user with either root access or full access to `ChatTwo` database.

UpdateDatabase

```
public static bool UpdateDatabase(string user, string password, string ip, int port)
{
    int cmdResult = 0;

    MySqlConnectionStringBuilder connBuilder = new MySqlConnectionStringBuilder();
    connBuilder.Add("User id", user);
    connBuilder.Add("Password", password);
    connBuilder.Add("Network Address", ip);
    connBuilder.Add("Port", port);
    //connBuilder.Add("Database", "ChatTwo");
    //connBuilder.Add("Connection timeout", 5);

    using (MySqlConnection tempConn = new MySqlConnection(connBuilder.ConnectionString))
    {
        // Get the database version number from the `ServerStatus` table.
        int version = -1;
        using (MySqlCommand cmd = new MySqlCommand("SELECT `ChatTwo`.`Version` FROM `ServerStatus`;", tempConn))
        {
            try
            {
                Open();
                // Execute SQL command.
                MySqlDataReader reader = cmd.ExecuteReader();
                if (reader.Read())
                {
                    version = (int)reader["Version"];
                }
            }
            finally
            {
                Close();
            }
        }

        switch (version)
        {
            case 0:
                throw new NotImplementedException("There is no update from version 0 (yet).");

                //using (MySqlCommand cmd = new MySqlCommand("UPDATE `ServerStatus` SET `Version` = 1, `LastUpdated` = NOW();", _conn))
                //{
                //    try
                //    {
                //        Open();
                //        // Execute SQL command.
                //        cmdResult = cmd.ExecuteNonQuery();
                //    }
                //    finally
                //    {
                //        Close();
                //    }
                //}
                //break;
            default:
                break;
        }
    }
    return (cmdResult != 0);
}
```

UpdateDatabase is for future backward compatibility. It may get a rewrite once I need it, but right now it simply gets the version number from the `ServerStatue` table. The version number is then used in a switch case to see what needs to be changed. The method returns true if any lines are affected by the update.

Connect

```
private static MySqlConnection _conn;

public static void Connect(string user, string password, string ip, int port)
{
    MySqlConnectionStringBuilder connBuilder = new MySqlConnectionStringBuilder();
    connBuilder.Add("User id", user);
    connBuilder.Add("Password", password);
    connBuilder.Add("Network Address", ip);
    connBuilder.Add("Port", port);
    connBuilder.Add("Database", "ChatTwo");
    connBuilder.Add("Connection timeout", 5);

    // Create the SqlConnection object using the saved IP address from settings.
    _conn = new MySqlConnection(connBuilder.ConnectionString);

    // Set the status of the database connection to on.
    _online = true;

    // Start the thread.
    _threadStatusIntervalUpdate = new Thread(() => StatusIntervalUpdate(connBuilder.ConnectionString));
    _threadStatusIntervalUpdate.Name = "StatusIntervalUpdate Thread (StatusIntervalUpdate method)";
    _threadStatusIntervalUpdate.Start();
}
```

The Connect method initializes the MySqlConnection and saves it as a private static object using the same parameters as the TestConnection method, creating a connection string with the MySqlConnectionStringBuilder system. It then starts the StatusIntervalUpdate looping method in another thread.

Disconnect

```
public static void Disconnect()
{
    // Set the status of the database connection to off.
    // This also makes the threaded method stop gracefully.
    _online = false;

    // Waits for the thread to end gracefully.
    if (_threadStatusIntervalUpdate != null)
        _threadStatusIntervalUpdate.Join();

    // Delete the SqlConnection object.
    _conn = null;
}
```

Disconnect is most of all used to gracefully stop the threaded StatusIntervalUpdate looping method.

Open and Close

```
private static int _SqlWorker = 0;

private static void Open()
{
    if (_SqlWorker == 0)
        _conn.Open();
    _SqlWorker++;
}

private static void Close()
{
    _SqlWorker--;
    if (_SqlWorker == 0 && _conn.State == ConnectionState.Open)
        _conn.Close();
}
```

Since I have a bit of multithreading going on with looping methods and so on, I attempted to make my Open and Close methods for the static MySqlConnection a little safer. It still needs some real stress testing. My backup solution to if this doesn't work is simply only have the connection string as a static variable and create a MySqlConnection everything I need to open the connection.

CreateUser

```
public static bool CreateUser(string username, string password)
{
    int cmdResult = 0;
    using (MySqlCommand cmd = new MySqlCommand("INSERT INTO `Users` (`Name`, `Password`) VALUES(@username, @password);", _conn))
    {
        // Add parameterized parameters to prevent SQL injection.
        cmd.Parameters.AddWithValue("@username", username);
        cmd.Parameters.AddWithValue("@password", password);

        try
        {
            Open();
            // Execute SQL command.
            cmdResult = cmd.ExecuteNonQuery();
        }
        catch (MySqlException ex)
        {
            if (ex.Number == 1062) // http://dev.mysql.com/doc/refman/5.6/en/error-messages-server.html
                // (ER_DUP_ENTRY) Message: Duplicate entry '%s' for key %d
                // If the username is already in use.
                return false;
            throw ex;
        }
        finally
        {
            Close();
        }
    }
    return (cmdResult != 0); // cmdResult content the number of affected rows.
}
```

CreateUser does was it says on the tin, it creates a new user with the desired username and password. The password has to be hashed and converted to a base64 string before being used in this method. The username has to be unique; if a user already exists with that username a MySqlException will be thrown and caught in the try-catch case.

ReadUser

```
static public UserObj ReadUser(int id)
{
    UserObj cmdResult = null;
    using (MySqlCommand cmd = new MySqlCommand("SELECT * FROM `Users` WHERE `ID` = @id;", _conn))
    {
        // Add parameterized parameters to prevent SQL injection.
        cmd.Parameters.AddWithValue("@id", id);

        try
        {
            Open();
            // Execute SQL command.
            MySqlDataReader reader = cmd.ExecuteReader();
            while (reader.Read())
            {
                cmdResult = new UserObj();
                cmdResult.ID = (int)reader["ID"];
                cmdResult.Name = (string)reader["Name"];
                cmdResult.Online = (bool)reader["Online"];
                cmdResult.StringSocket(reader["Socket"].ToString());
                cmdResult.LastOnline = (DateTime)reader["LastOnline"];//, _ci);
                cmdResult.Registered = (DateTime)reader["Registered"];//, _ci);
            }
        }
        finally
        {
            Close();
        }
    }
    return cmdResult;
}
```

This method was mostly made for my early testing of the database. Parts of it may be used later once I get more of the server application done.

LoginUser

```
static public UserObj LoginUser(string name, string password)
{
    UserObj cmdResult = null;
    using (MySqlCommand cmd = new MySqlCommand("SELECT `ID`, `Name` FROM `Users` WHERE `Name` = @name AND `Password` = @password;", _conn))
    {
        // Add parameterized parameters to prevent SQL injection.
        cmd.Parameters.AddWithValue("@name", name);
        cmd.Parameters.AddWithValue("@password", password);

        try
        {
            Open();
            // Execute SQL command.
            MySqlDataReader reader = cmd.ExecuteReader();
            while (reader.Read())
            {
                cmdResult = new UserObj();
                cmdResult.ID = (int)reader["ID"];
                cmdResult.Name = reader["Name"].ToString();
            }
        }
        finally
        {
            Close();
        }
    }
    return cmdResult;
}
```

The LoginUser method is used to validate a login attempt by simply querying for a username and password match. If a match exist, return the ID of the user so further work can be done for the newly logged in user. If a match doesn't exist, return null.

StatusIntervalUpdate

```
static public void StatusIntervalUpdate(string connString) // Threaded looping method.
{
    using (MySqlConnection intervalConn = new MySqlConnection(connString))
    {
        MySqlCommand cmd = new MySqlCommand("StatusIntervalUpdate", intervalConn);
        //Set up cmd to reference stored procedure 'StatusIntervalUpdate'.
        cmd.CommandType = System.Data.CommandType.StoredProcedure;

        try
        {
            while (_online)
            {
                intervalConn.Open();
                // Execute SQL command.
                MySqlDataReader reader = cmd.ExecuteReader();
                while (reader.Read())
                {
                    int userId = (int)reader["ID"];
                    Thread userStatusChange = new Thread(() => UserStatusChanged(userId, false));
                    userStatusChange.Name = "UserChange Thread (UserStatusChanged method)";
                    userStatusChange.Start();
                }
                intervalConn.Close();
                Thread.Sleep(1000); // 1 seconds.
            }
        }
        catch (Exception ex)
        {
            System.Diagnostics.Debug.WriteLine("### " + _threadStatusIntervalUpdate.Name + " has crashed:");
            System.Diagnostics.Debug.WriteLine("### " + ex.Message);
            System.Diagnostics.Debug.WriteLine("### " + ex.ToString());
        }
        finally
        {
            if (intervalConn.State == ConnectionState.Open)
                intervalConn.Close();
        }
    }
}
```

Once started, this threaded looping method will keep looping once every second while _online remains true. The method has its own MySqlConnection object, this is to prevent the static MySqlConnection from being used too much.

The method uses the StatusIntervalUpdate stored procedure. As explained in the Database section of this report, it makes a query for all users that are online and have a `LastOnline` timestamp that is older than 10 seconds and returns them before changing them to offline. This method that starts a single run thread of the UserStatusChanged method for each returned user.

ChatTwo Protocol

The ChatTwo Protocol, while not totally finished yet, handles all the communication between server, client, and other clients. The server-side of the protocol handles user's login and online status. Meanwhile the client-side of the protocol is more focused on keeping alive the connection to the server and messaging other clients.

Login

When a client attempts to log in, it constructs a "Login" message. This message contents a hash of the entered password and the entered username. The server then does a normal select query looking for a match on both the username and the hashed password. If a match is found the server replies back with the user ID of that user in a LoginReply message. If there is no match for the username and password, the server throws a fit because I had a little problem with authenticating a failed login message with my current user authentication system.

LoginReply

If the client receive a successful LoginReply message, it will save the user ID it receive as its own and start using it to identify itself. This allow for future messages to be easily authenticated, because the receiving server/client don't have to test with all the possible users' shared secrets.

Status

The client then begin to send periodic Status messages to the server as a form of keepalive, this also keeps the server up-to-date with the IPEndPoint it can reach the client on, and at the same time doing UDP hole punching through the client's gateway NAT router. Each time the server receive a Status message, it updates a user's "Last Online" entry on the database.

ContactStatus

The server then executes the StatusIntervalUpdate stored procedure periodically, which returns and sets all users that have a `LastOnline` date that is older than 10 seconds to detect user timeouts.

Then each user that is changed to "`Online` = 0" is then checked if they have any contacts that is online and send each of the online contacts an ContactStatus message about the user going offline.

The server does not currently send out ContactStatus messages to the user's contacts to tell them that the user has come online when they successfully login. Most of the code is there, it is just not fully implemented yet. It is also currently not possible to add a contact, so there was no one to send the messages to anyway.

Contact-related

All the contact related messages haven't been implemented yet.

Message

Message messages also haven't been implemented yet.

Logout

Logout messages are planned but not fully thought out yet.

User Authentication

When I got to the question of how to know if a user is logged in or not, I asked our networking teacher Bent for some quick advice. Bent pointed out that a MAC (message authentication code) could be the solution.

With my head full of hashing and shared secrets, I went to code. I quickly ran into the fun problem of all of the MAC checks failing horribly without reason.

Further testing showed that it was not me that didn't do it correct. Microsoft just doesn't know what a hash is, or maybe it was too hard to implement so they left it broken. This is what I found:

```
byte[] messageOne = new byte[] { 0x92, 0xFF, 0xDE, 0xA2 };
byte[] messageTwo = new byte[] { 0x92, 0xFF, 0xDE, 0xA2 };
bool result = (messageOne.GetHashCode() == messageTwo.GetHashCode());
```

That piece of code will always return false because ".GetHashCode()" does not work on objects and arrays. Rather than base the integer that it return on the content of the object, it returns an almost random number because it is only based on where object is saved in the memory or something like that.

Solution

I found a way to make do some real hashing by searching online, some people had had that problem before. So this is what I ended up adding to my ByteHelper class:

```
static public string GetHashString(byte[] bytes)
{
    string hash = Convert.ToBase64String(GetHashBytes(bytes));
    return hash;
}

static public byte[] GetHashBytes(byte[] bytes)
{
    byte[] hash;
    using (SHA1CryptoServiceProvider sha1 = new SHA1CryptoServiceProvider())
    {
        hash = sha1.ComputeHash(bytes);
    }
    return hash;
}
```

Also learned how to use Base64 strings which made it much easier to both look at and debug.

Now with working hashing methods I constructed a system for creating and validating MACs.

```
public static bool ValidateMac(byte[] bytes, string sharedSecret)
{
    string mac = Convert.ToBase64String(bytes, 2, ByteHelper.HashByteLength);
    bool macValid = CreateMac(ByteHelper.SubArray(bytes, SignatureByteLength + ByteHelper.HashByteLength), sharedSecret) == mac;
    return macValid;
}

public static byte[] AddSignatureAndMac(byte[] bytes, string sharedSecret)
{
    TimeSpan sinceMidnight = DateTime.Now - DateTime.Today;
    int timez = (int)sinceMidnight.TotalMilliseconds;
    bytes = ByteHelper.ConcatinateArray(BitConverter.GetBytes(timez), bytes); // Add a milisecond timestamp to the meassage.

    byte[] macBytes = Convert.FromBase64String(CreateMac(bytes, sharedSecret));

    byte[] singatureBytes = new byte[] { 0x92, _version }; // Signature byte and version byte.

    bytes = ByteHelper.ConcatinateArray(singatureBytes, macBytes, bytes);
    return bytes;
}
```

It took some trial and error to get to this final version, but it works nicely. The only thing that is then needed is a shared secret that both the client and server need to know or create from the same source.

Shared Secret

I decided to create the shared secret from the user's login request message. This worked nicely until I got to the point where I had to plan on how to send back messages with "login failed" messages.

I never got to implement client to client communication, and one of the reasons for that is that I haven't decided on what the shared secret should be between them. The easiest way seems to be to have the server simply make a shared secret from the two users' shared secrets they have with the server.

Unique Hash

To make sure that all messages have a unique hash, I have added a millisecond timestamp to be part of the message before the hashing. This helps when multiple identical messages have to be sent, because without the timestamp sending the same message two times in a row would make the second message be treated as a duplicate and dropped.

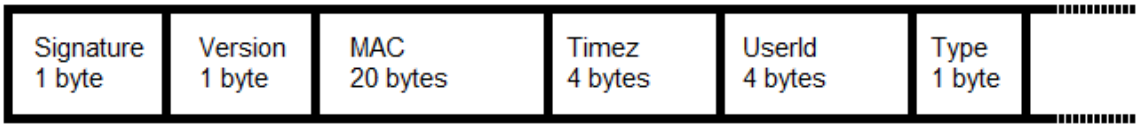
Signature

Along with the MAC, I also have a signature that I use to identify if the message is even for my program. I use 0x92 because it is my birthday, the 9th of February. There is also a byte I have reserved as a version number. This will be used in case of future protocol changes to allow backward compatibility.

Message Syntax

The byte syntax for each message type is important to lay out. Since Both ends of the communication need to know where all the correct pieces of data and information is stored.

Header



The header is of course the most important of them all. This is because it is added in front of all messages.

In my header I have first of all the Signature byte which is always 0x92 for this project. This simply makes my program able to detect if the message is indeed meant for it, or if another program has accidently send an unrelated message to it.

Version byte is reserved for future backward compatibility.

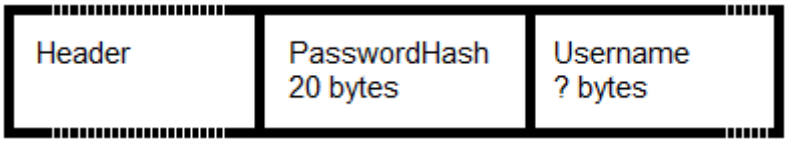
MAC is the message authentication code, used to validate the sender and integrity of the message.

Timez is the millisecond timestamp since midnight.

UserId is the ID number of the sending user.

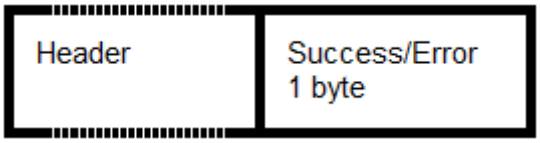
The last part of header is the Type byte, the value is determined by an enum in the ChatTwo_Protocol class.

CreateUser



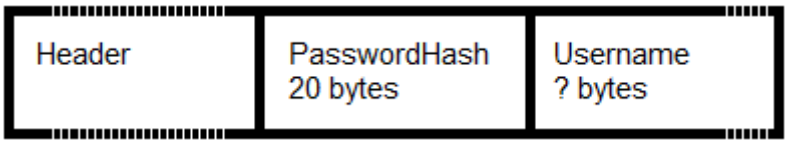
The CreateUser message has two pieces of information. PasswordHash is 20 bytes long and contain the SHA1 hash of the user's password. Username is the Unicode encoding of the username.

CreateUserReply



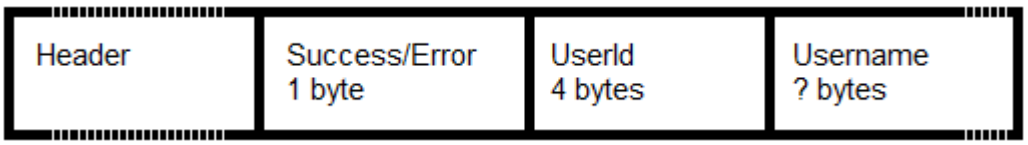
The CreateUserReply only has one byte that contains an error number, if this number is 0x00 there is no error.

Login



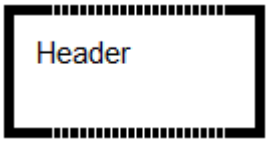
A Login message contains PasswordHash and Username just like CreateUser.

LoginReply



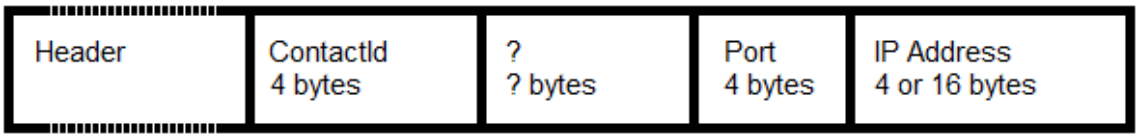
LoginReply Contains an error number much like CreateUserReply, but if there is no error the message also contains the UserId and the username of the now logged in user.

Status



Status messages are empty. There is no need to add anything after the header because the only information that is needed is the source IP address and port number.

ContactStatus



The ContactStatus message is not fully planned out yet; I know I want it to contain UserId, port number, and IP address of the contact. But I know I will want it to contain more information, I am just not totally sure how to structure it yet.

Remaining Message Types

The remaining messages types are yet to be totally planned out, but should be somewhat simple.

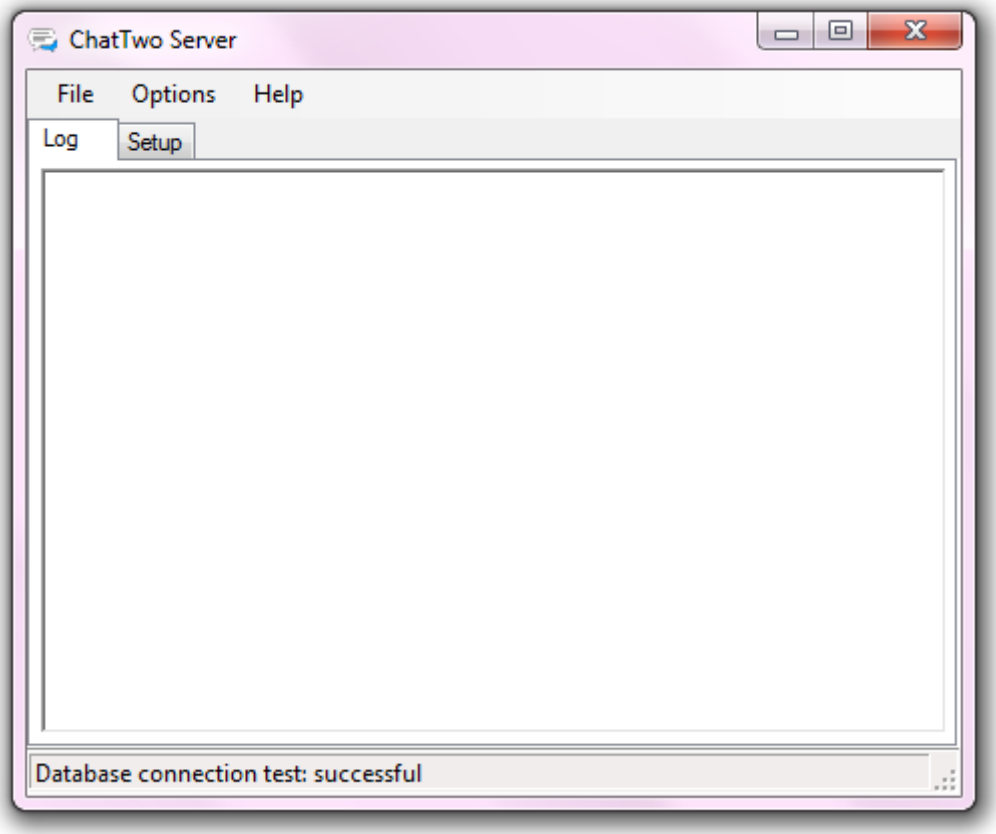
User Interface

The user interface is very much subject to change, for example the client interface is pretty much temporary.

Server Interface

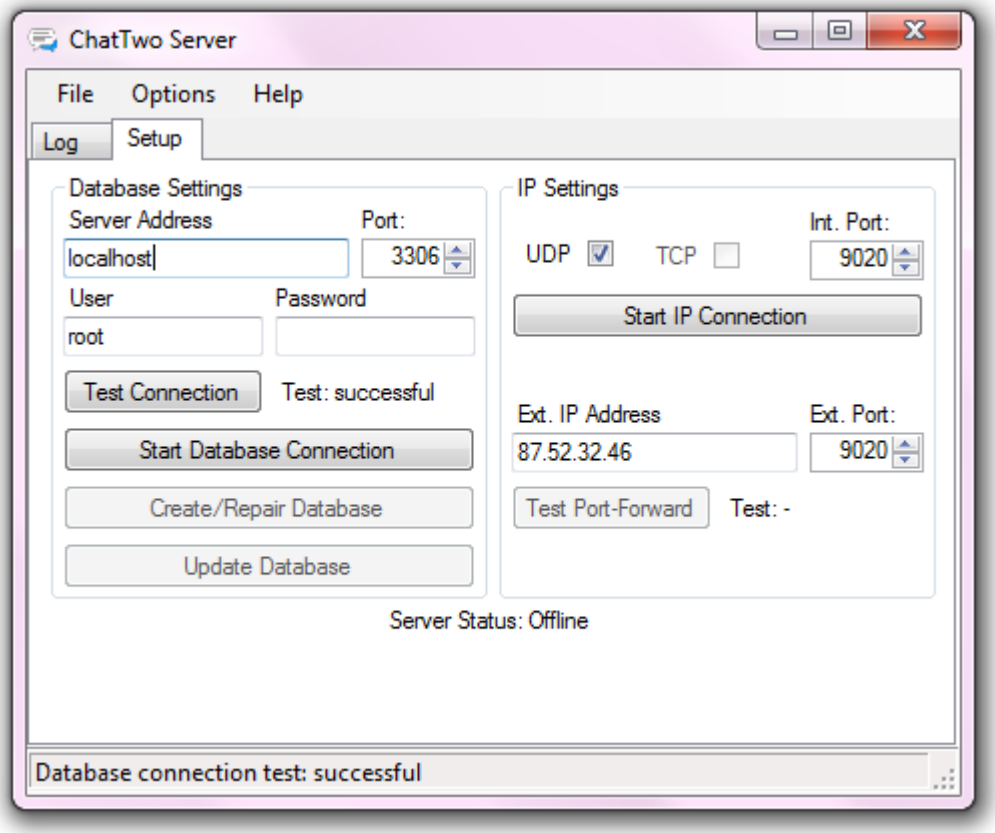
The server application's user interface is a design that I have used in one of my other programs. I wouldn't say it is all that good, but it works for more.

Log



The Log tab is supposed to show events and debug messages. Only the debug messages for SQL test commends are implemented so far.

Setup



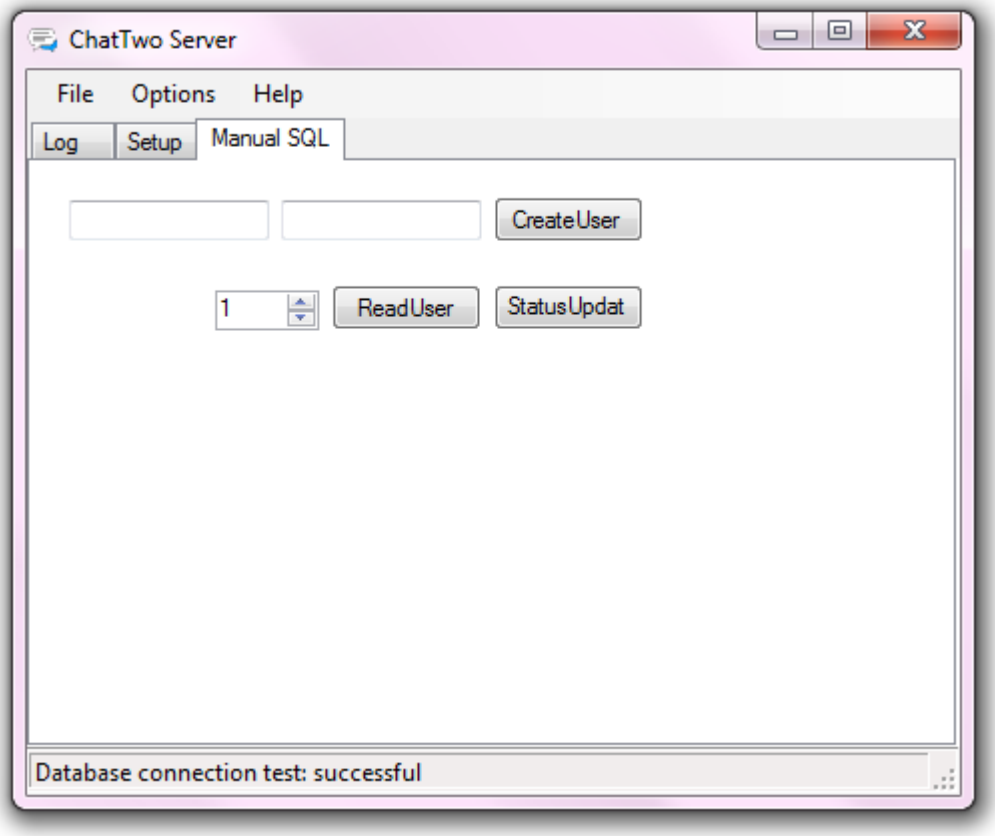
The Setup tab is used to setup the server. Here you configure the MySQL server's connection string and network socket.

Before a database connection can be established, the entered connection data need to be tested. This is done by click the "Test Connection" button. If the test fails, an error message will be displayed with some information that may be of use. If the connection test to the MySQL server is successfully established, but the database or a table on the database doesn't exist, the "Create/Repair Database" button will become available. The "Update Database" button is for future updates to allow for updating of older databases. If the connection test is successful without error, the "Start Database Connection" button is enabled. When the "Start Database Connection" button is pressed, the DatabaseCommunication's Connect method is executed with the connection information entered above.

The IP setup is a little simpler. You can select the protocol to use, but only UDP is possible at the time. The local port number of the server can be selected from the NumericUpDown control, when "Start IP Connection" button is clicked the UdpClient is started on that port

number. Once the IP connection is setup, it is possible to test if the server is accessible from on the external IP address and port.

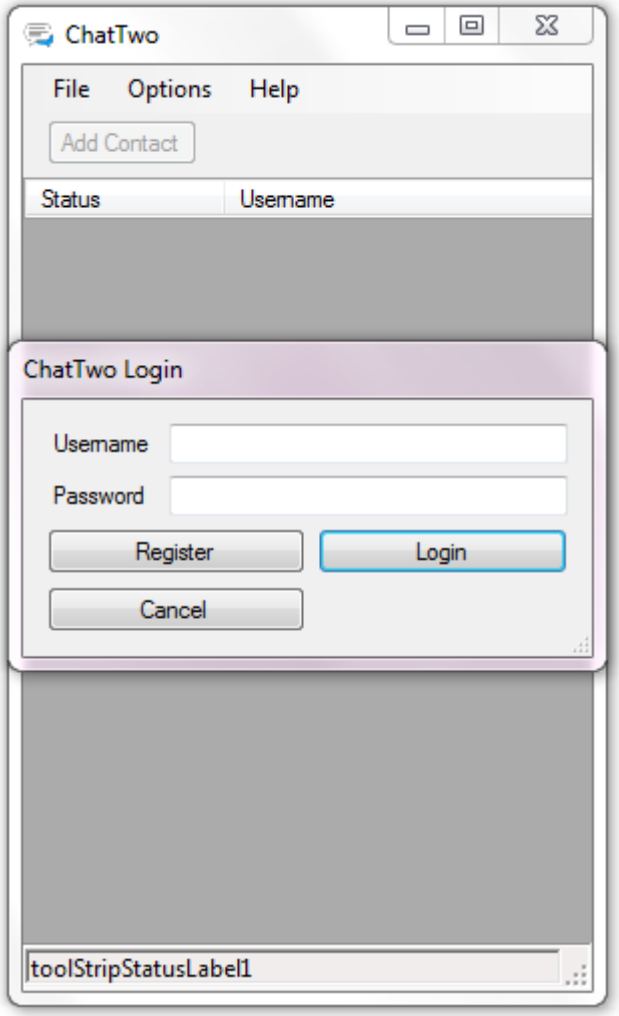
Manual SQL



The Manual SQL tab is most of all used to test the database. It can create users, filling in a username and password in the two textboxes respectively and clicking the “CreateUser” button.

The “ReadUser” and “StatusUpdate” buttons queries the database using the NumericUpDown control’s value as the UserId.

Client Interface



The client interface is very simple and most of it is not implemented. The only things that work is the “File” menu strip items: “Login”, “Logout”, and “Close”. Clicking “Login” brings up the login dialog, allowing the user to login or create a new user with the “Register” option.

Tests

No large scale tests have taken place since most of the core program still needs to be implemented.

What I have tested:

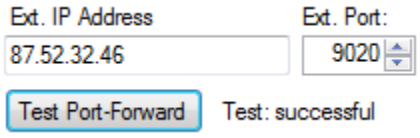
1. Database error detection.
2. Database creation.
3. Self UDP port forward test.
4. UDP hole punching.
5. User creation.
6. User login.

Results

1. Server is able to detect if the database doesn't exist and if there are missing tables in the database.

   Database connection test: failed. One or more of the tables do not exist

2. Server can create the database from scratch or replace missing tables without deleting the existing tables.
3. Server is able to test the port forward.



4. Clients are able to communicate with the server using UDP through NAT routers with no sign of trouble.
5. Clients are able to create users remotely on the server.
6. Clients can login with created user and keep the connection alive with the Status keepalive messages.

```
18:49:35 user[1] Name: Admin
         user[1] Online: True
         user[1] Socket: 87.52.32.46:64190
         user[1] LastOnline: 2015-01-01 18:49:35
         user[1] Registered: 2015-01-01 17:42:13
18:50:02 user[1] Name: Admin
         user[1] Online: True
         user[1] Socket: 87.52.32.46:64190
         user[1] LastOnline: 2015-01-01 18:50:02
         user[1] Registered: 2015-01-01 17:42:13
```

Conclusion

Conclusion on the Problem Formulation

Making something as good as one of the mainstream chat programs will take a lot longer and more knowledge about C# programming than I currently have. I would likely be able to make the ChatTwo fully functional if a couple of months, but it would in no way look as nice as the mainstream products.

When researching how common peer-to-peer applications got through NAT routers, I learned about “Hole Punching”. This technic easily answered all my questions about how to communicate through NAT routers, as well as being very easy to implement.

The minimum security that I found to be necessary was a form of user authentication to ensure that once logged in, the server can recognize that it is the same client that is talk with the server. Other than base it on the clients IPEndPoint, a message authentication code (MAC) makes me able to have connections with clients that change IP address or port number on the fly.

Knowing how to manually setup port-forwarding on my home router, the idea of actually having a server simply became a question of paying the electricity bill while my desktop is turned on 24/7 and maybe acquire a static IP address from my ISP.

Personal Conclusion

There are still lots of work to be done before ChatTwo is finished. But it is getting along and I'd expect it only to take me a couple of months to have it fully working.

Apart from all the work in progress features, everything seems to work so far. There are a few areas that I would like to redo though. For example; I want to try and make the server track user online status part of the server application rather than the SQL server.
One possibly good thing about using the SQL server to track user online status, would be that multiple server applications can manage the same system. But then I will have to think of another way to track user timeouts anyway.

There are also a ton of smaller holes in my current design. For example usernames are case sensitive and allow for creation of users with the same name but diffident case. Like "Deantwo" and "deantwo" can both exist at the same time.

If I had maybe also read a bit more about existing chat program protocols before jumping head first into the coding. I could probably have saved myself from some bad headaches and long nights. But some of the fun about programing is failing a little and then finding a solution.

List of references

- https://www.apachefriends.org/index.html (2014-11-26)

  XAMPP is the most popular PHP development environment

- https://github.com/ (2014-12-19)

  GitHub is a free web-based Git repository hosting service.

Appendix

To make view of my code easier and even allow the reader to test it themselves. You can view and even download the entire project from GitHub. I made a separate branch with the version of the code displayed in this report here:

- https://github.com/Deantwo/ChatTwo-Server/tree/Report
- https://github.com/Deantwo/ChatTwo/tree/Report

Server Code

FormMain.cs

```
001 using System;
002 using System.Collections.Generic;
003 using System.ComponentModel;
004 using System.Data;
005 using System.Drawing;
006 using System.Linq;
007 using System.Text;
008 using System.Windows.Forms;
009
010 namespace ChatTwo_Server
011 {
012     public partial class FormMain : Form
013     {
014         UdpCommunication _server;
015
016         Image _infoTip;
017
018         public FormMain()
019         {
020             InitializeComponent();
021             Global.MainWindow = this; // This is just so I can
call WriteLog from other classes.
022
023 #if DEBUG
024             this.Text += " (DEBUG)";
025 #endif
026
027             _server = new UdpCommunication();
028             _server.PacketReceived +=
ChatTwo_Server_Protocol.MessageReceivedHandler;
029             ChatTwo_Server_Protocol.MessageTransmission +=
_server.SendPacket;
030             DatabaseCommunication.UserStatusChange +=
ChatTwo_Server_Protocol.TellUserAboutContactstatusChange;
031             _server.EtherConnectionReply += EtherConnectReply;
032
033             notifyIcon1.BalloonTipTitle = this.Text;
034             notifyIcon1.Text = this.Text;
```

```
035             notifyIcon1.Icon =
Icon.ExtractAssociatedIcon(Application.ExecutablePath);
036             this.Icon =
Icon.ExtractAssociatedIcon(Application.ExecutablePath);
037
038             // Steal the system "informaion icon" and resize it.
039             _infoTip = SystemIcons.Information.ToBitmap();
040             _infoTip = (Image)(new Bitmap(_infoTip, new Size(12,
12)));
041
042             tabSqlTest.Parent = null;
043         }
044
045         #region Database Setup
046         private void tbxSql_ConnectionStringValuesChanged(object
sender, EventArgs e)
047         {
048             btnSqlConnect.Enabled = false;
049             btnSqlCreate.Enabled = false;
050             btnSqlUpdate.Enabled = false;
051             lblSqlConnection.Text = "Test: -";
052             lblSqlConnection.Image = null;
053         }
054
055         private void btnSqlTest_Click(object sender, EventArgs e)
056         {
057             tbxSql_ConnectionStringValuesChanged(null, null);
058
059             // Basic user feedback.
060             toolStripStatusLabel1.Text = "Testing connection and
access to the database...";
061             statusStrip1.Refresh(); // Has to be done or the
statusStrip won't display the new toolStripStatusLabel text.
062
063             // Test the connection and access, giving the user
feedback if it fails.
064             DatabaseCommunication.ConnectionTestResult dbTest =
DatabaseCommunication.TestConnection(tbxSqlUser.Text,
tbxSqlPassword.Text, tbxSqlAddress.Text, (int)nudSqlPort.Value);
065
066             // Check the result of the connetion test.
067             bool connectTestSuccessful = dbTest ==
DatabaseCommunication.ConnectionTestResult.Successful;
068             if (connectTestSuccessful)
069             {
070                 toolStripStatusLabel1.Text = "Database connection
test: successful";
071                 lblSqlConnection.Text = "Test: successful";
072                 toolTip1.SetToolTip(lblSqlConnection, "");
073                 btnSqlConnect.Enabled = true;
074             }
075             else
076             {
077                 string errorMessage;
078                 string errorTip;
079                 switch (dbTest)
080                 {
```

```
081                     case
DatabaseCommunication.ConnectionTestResult.NoConnection:
082                         errorMessage = "Could not connect to the
server at \"" + tbxSqlAddress.Text + ":" + (int)nudSqlPort.Value +
"\"";
083                         errorTip = "." + Environment.NewLine +
Environment.NewLine +
084                             "Please make sure the MySQL server is
running and the IP address and port is correct.";
085                         break;
086                     case
DatabaseCommunication.ConnectionTestResult.FailLogin:
087                         errorMessage = "Login rejected by server";
088                         errorTip = "." + Environment.NewLine +
Environment.NewLine +
089                             "Please make sure the username and
password is correct.";
090                         break;
091                     case
DatabaseCommunication.ConnectionTestResult.NoPermission:
092                         errorMessage = "Permission failed for
user";
093                         errorTip = "." + Environment.NewLine +
Environment.NewLine +
094                             "Talk to the system admin to gain
access to the MySQL server.";
095                         break;
096                     case
DatabaseCommunication.ConnectionTestResult.MissingDatabase:
097                         errorMessage = "Database does not exist";
098                         errorTip = "." + Environment.NewLine +
Environment.NewLine +
099                             "You can create/repair the database by
clicking the button below.";
100                         btnSqlCreate.Enabled = true;
101                         break;
102                     case
DatabaseCommunication.ConnectionTestResult.MissingTable:
103                         errorMessage = "One or more of the tables
do not exist";
104                         errorTip = "." + Environment.NewLine +
Environment.NewLine +
105                             "You can create/repair the database by
clicking the button below.";
106                         btnSqlCreate.Enabled = true;
107                         break;
108                     case
DatabaseCommunication.ConnectionTestResult.OutDated:
109                         errorMessage = "SQL database need to be
updated";
110                         errorTip = ".";
111                         btnSqlUpdate.Enabled = true;
112                         break;
113                     default:
114                         errorMessage = "Unknown SQL error";
115                         errorTip = ".";
116                         break;
117                 }
```

```
118                 MessageBox.Show(errorMessage + errorTip, "SQL
Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
119                 toolStripStatusLabel1.Text = "Database connection
test: failed. " + errorMessage;
120                 lblSqlConnection.Text = "Test: failed";
121                 lblSqlConnection.Image = _infoTip;
122                 toolTip1.SetToolTip(lblSqlConnection, errorMessage
+ errorTip);
123             }
124         }
125
126         private void btnSqlConnect_Click(object sender, EventArgs
e)
127         {
128             tbxSqlUser.ReadOnly = !tbxSqlUser.ReadOnly;
129             tbxSqlPassword.ReadOnly = !tbxSqlPassword.ReadOnly;
130             tbxSqlAddress.ReadOnly = !tbxSqlAddress.ReadOnly;
131             nudSqlPort.Enabled = !nudSqlPort.Enabled;
132             btnSqlTest.Enabled = !btnSqlTest.Enabled;
133             if (DatabaseCommunication.Active)
134             {
135                 DatabaseCommunication.Disconnect();
136                 btnSqlConnect.Text = "Start Database Connection";
137                 tabSqlTest.Parent = null;
138             }
139             else
140             {
141                 DatabaseCommunication.Connect(tbxSqlUser.Text,
tbxSqlPassword.Text, tbxSqlAddress.Text, (int)nudSqlPort.Value);
142                 btnSqlConnect.Text = "Stop Database Connection";
143                 tabSqlTest.Parent = tabControl1;
144             }
145             UpdateServerStatus();
146         }
147
148         private void btnSqlCreate_Click(object sender, EventArgs
e)
149         {
150             bool worked =
DatabaseCommunication.CreateDatabase(tbxSqlUser.Text,
tbxSqlPassword.Text, tbxSqlAddress.Text, (int)nudSqlPort.Value);
151             if (worked)
152             {
153                 tbxSql_ConnectionStringValuesChanged(null, null);
154                 toolStripStatusLabel1.Text = "SQL database
created/repaired";
155             }
156             else
157                 WriteLog("Could not create the database.",
Color.Red.ToArgb());
158         }
159
160         private void btnSqlUpdate_Click(object sender, EventArgs
e)
161         {
162             bool worked =
DatabaseCommunication.UpdateDatabase(tbxSqlUser.Text,
tbxSqlPassword.Text, tbxSqlAddress.Text, (int)nudSqlPort.Value);
```

```
163             if (worked)
164             {
165                 tbxSql_ConnectionStringValuesChanged(null, null);
166                 toolStripStatusLabel1.Text = "SQL database
Updated";
167             }
168             else
169                 WriteLog("Could not update the database.",
Color.Red.ToArgb());
170         }
171         #endregion
172
173         #region IP Setup
174         private void chxIp_CheckedChanged(object sender, EventArgs
e)
175         {
176             btnIpConnect.Enabled = chxIpUdp.Checked ||
chxIpTcp.Checked;
177         }
178
179         private void btnIpConnect_Click(object sender, EventArgs
e)
180         {
181             nudIpPort.Enabled = !nudIpPort.Enabled;
182             btnIpTest.Enabled = !btnIpTest.Enabled;
183             chxIpUdp.Enabled = !chxIpUdp.Enabled;
184             //chxIpTcp.Enabled = !chxIpTcp.Enabled; // Not
implemented yet.
185             if (_server.Active)
186             {
187                 _server.Stop();
188                 btnIpConnect.Text = "Start IP Connection";
189             }
190             else
191             {
192                 _server.Start((int)nudIpPort.Value);
193                 btnIpConnect.Text = "Stop IP Connection";
194             }
195             UpdateServerStatus();
196         }
197
198         private void tbxIp_ExternalIpValuesChanged(object sender,
EventArgs e)
199         {
200             lblIpConnection.Text = "Test: -";
201             lblIpConnection.Image = null;
202         }
203
204         private void btnIpTest_Click(object sender, EventArgs e)
205         {
206             tbxIp_ExternalIpValuesChanged(null, null);
207
208             if (chxIpUdp.Checked)
209             {
210                 // Basic user feedback.
211                 toolStripStatusLabel1.Text = "Testing UDP port-
forward...";
```

```
212                 statusStrip1.Refresh(); // Has to be done or the
statusStrip won't display the new toolStripStatusLabel text.
213
214                 string errorMessage;
215                 string errorTip;
216                 System.Net.IPAddress address;
217                 if
(System.Net.IPAddress.TryParse(tbxIpExternalAddress.Text, out
address))
218                 {
219                     // Check the result of the port-forward test.
220                     bool portforwardTestStartSuccessful =
UdpCommunication.TestPortforward(new System.Net.IPEndPoint(address,
(int)nudIpExternalPort.Value));
221                     if (portforwardTestStartSuccessful)
222                     {
223                         lblIpConnection.Text = "Test: testing...";
224                         toolTip1.SetToolTip(lblIpConnection, "");
225                         timer1.Start();
226                         return;
227                     }
228                     else
229                     {
230                         errorMessage = "The UDP port-forward test
failed";
231                         errorTip = "." + Environment.NewLine +
Environment.NewLine +
232                             "Could not start the test. Please
ensure you have an internet connection.";
233                         lblIpConnection.Text = "Test: failed";
234                     }
235                 }
236                 else
237                 {
238                     errorMessage = "The entered external IP
address is invalid";
239                     errorTip = "." + Environment.NewLine +
Environment.NewLine +
240                         "Please enter a valid IP address." +
Environment.NewLine +
241                         "If you don't know your external IP
address, you can get it by googling \"what is my IP?\".";
242                     lblIpConnection.Text = "Test: invalid IP
address";
243                 }
244
245                     MessageBox.Show(errorMessage + errorTip,
"Port-Forward Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
246                 toolStripStatusLabel1.Text = "UDP port-forward
test: failed. " + errorMessage;
247                     lblIpConnection.Image = _infoTip;
248                     toolTip1.SetToolTip(lblIpConnection, errorMessage
+ errorTip);
249             }
250             if (chxIpTcp.Checked)
251             {
252                 // Basic user feedback.
```

```
253                 toolStripStatusLabel1.Text = "Testing TCP port-
forward...";
254                 statusStrip1.Refresh(); // Has to be done or the
statusStrip won't display the new toolStripStatusLabel text.
255
256                 throw new NotImplementedException("TCP is not
implemented yet.");
257             }
258         }
259
260         public void EtherConnectReply(object sender, EventArgs
args)
261         {
262             if (lblIpConnection.InvokeRequired)
263             { // Needed for multi-threading cross calls.
264                 this.Invoke(new Action<object,
EventArgs>(this.EtherConnectReply), new object[] { sender, args });
265             }
266             else
267             {
268                 if (timer1.Enabled)
269                 {
270                     timer1.Stop();
271                     toolStripStatusLabel1.Text = "UDP port-forward
test: successful";
272                     lblIpConnection.Text = "Test: successful";
273                     toolTip1.SetToolTip(lblIpConnection, "");
274                 }
275             }
276         }
277
278         private void timer1_Tick(object sender, EventArgs e)
279         {
280             if (timer1.Enabled)
281             {
282                 timer1.Stop();
283                 string errorMessage;
284                 string errorTip;
285
286                 errorMessage = "The UDP port-forward test failed";
287                 errorTip = "." + Environment.NewLine +
Environment.NewLine +
288                     "Please ensure your router is correctly
configured.";
289                 lblIpConnection.Text = "Test: failed";
290
291                 MessageBox.Show(errorMessage + errorTip, "Port-
Forward Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
292                 toolStripStatusLabel1.Text = "UDP port-forward
test: failed. " + errorMessage;
293                 lblIpConnection.Image = _infoTip;
294                 toolTip1.SetToolTip(lblIpConnection, errorMessage
+ errorTip);
295             }
296         }
297         #endregion
298
299         private void UpdateServerStatus()
```

```
300         {
301             lblServerStatus.Text = "Server Status: ";
302             if (!_server.Active)
303                 lblServerStatus.Text += "Offline";
304             else if (!DatabaseCommunication.Active)
305                 lblServerStatus.Text += "No database connection";
306             else
307                 lblServerStatus.Text += "Online";
308         }
309
310         public void WriteLog(string log, int colorARGB = -
16777216) // 0xFF000000 (Color.Black)
311         {
312             if (rtbxLog.InvokeRequired)
313             { // Needed for multi-threading cross calls.
314                 this.Invoke(new Action<string,
int>(this.WriteLog), new object[] { log, colorARGB });
315             }
316             else
317             {
318                 // Add timestamp to the log entry.
319                 string timestamp =
DateTime.Now.ToString("HH:mm:ss"); // "yyyy-MM-dd HH:mm:ss"
320                 log = timestamp + " " + log;
321                 // And this part only matters for the "### Start\n
..." message.
322                 if (log.Contains(Environment.NewLine))
323                 {
324                     int i = timestamp.Length;
325                     timestamp = "";
326                     for (; i > 0; i--)
327                         timestamp += " ";
328                     log = log.Replace(Environment.NewLine,
Environment.NewLine + timestamp + " ");
329                 }
330
331                 // Just to prevent the first line from being
empty.
332                 if (rtbxLog.Text != String.Empty)
333                     rtbxLog.AppendText(Environment.NewLine);
334
335                 int lengthBeforeAppend = rtbxLog.Text.Length;
336                 // Write log to the textbox.
337                 rtbxLog.AppendText(log);
338                 // Put the focus on the textbox.
339                 rtbxLog.Focus();
340                 // Color the text.
341                 rtbxLog.SelectionStart = lengthBeforeAppend;
342                 rtbxLog.SelectionLength = log.Length;
343                 rtbxLog.SelectionColor =
Color.FromArgb(colorARGB);
344
345                 // Delete the top line when there is over 1000
lines.
346                 if (rtbxLog.Lines.Length > 1000)
347                 {
348                     rtbxLog.SelectionStart = 0;
```

```
349                     rtbxLog.SelectionLength =
rtbxLog.GetFirstCharIndexFromLine(1);
350                     rtbxLog.ReadOnly = false; // Can't edit the
text when the RichTextBox is in ReadOnly mode.
351                     rtbxLog.SelectedText = String.Empty;
352                     rtbxLog.ReadOnly = true;
353                 }
354
355                 // Put the cursor at the end of the text.
356                 rtbxLog.Select(rtbxLog.Text.Length, 0);
357                 // Scroll to the bottom of the textbox.
358                 rtbxLog.ScrollToCaret();
359             }
360         }
361
362         #region Database Test Commands
363         private void CreateUser_Click(object sender, EventArgs e)
364         {
365             string hashedPassword =
ByteHelper.GetHashString(Encoding.Unicode.GetBytes(textBox2.Text));
366             bool worked =
DatabaseCommunication.CreateUser(textBox1.Text, hashedPassword);
367             if (worked)
368                 WriteLog("user[\"" + textBox1.Text + "\"] Was
created successfully.", Color.Blue.ToArgb());
369             else
370                 WriteLog("user[\"" + textBox1.Text + "\"] Was not
created.", Color.Red.ToArgb());
371         }
372
373         private void ReadUser_Click(object sender, EventArgs e)
374         {
375             UserObj user =
DatabaseCommunication.ReadUser((int)numericUpDown1.Value);
376             if (user != null)
377                 WriteLog(user.ToString(), Color.Purple.ToArgb());
378             else
379                 WriteLog("user[" + (int)numericUpDown1.Value + "]
Does not exist.", Color.Red.ToArgb());
380         }
381
382         private void StatusUpdate_Click(object sender, EventArgs
e)
383         {
384             bool worked =
DatabaseCommunication.UpdateUser((int)numericUpDown1.Value, new
System.Net.IPEndPoint(new System.Net.IPAddress(new byte[] { 10, 0, 0,
1} ), 9020));
385             if (worked)
386                 WriteLog("user[" + (int)numericUpDown1.Value + "]
Changed to online.", Color.Blue.ToArgb());
387             else
388                 WriteLog("user[" + (int)numericUpDown1.Value + "]
Does not exist.", Color.Red.ToArgb());
389         }
390         #endregion
391
392         #region Options
```

```
393         private void minimizeToTrayToolStripMenuItem_Click(object
sender, EventArgs e)
394         {
395             minimizeToTrayToolStripMenuItem.Checked =
!minimizeToTrayToolStripMenuItem.Checked;
396             //Properties.Settings.Default.setTray =
minimizeToTrayToolStripMenuItem.Checked;
397             //Properties.Settings.Default.Save();
398         }
399
400         private void showPasswordsToolStripMenuItem_Click(object
sender, EventArgs e)
401         {
402             showPasswordsToolStripMenuItem.Checked =
!showPasswordsToolStripMenuItem.Checked;
403             tbxSqlPassword.UseSystemPasswordChar =
!showPasswordsToolStripMenuItem.Checked;
404         }
405         #endregion
406
407         #region Minimize to Tray
408         private void FormMain_Resize(object sender, EventArgs e)
409         {
410             if (minimizeToTrayToolStripMenuItem.Checked)
411             {
412                 if (FormWindowState.Minimized == this.WindowState)
413                 {
414                     TrayBalloonTip("Minimized to tray",
ToolTipIcon.None);
415                     this.ShowInTaskbar = false;
416                 }
417                 else if (FormWindowState.Normal ==
this.WindowState)
418                 {
419                     this.ShowInTaskbar = true;
420                 }
421             }
422         }
423
424         private void notifyIcon1_MouseDoubleClick(object sender,
MouseEventArgs e)
425         {
426             this.WindowState = FormWindowState.Normal;
427         }
428
429         private void RestoreToolStripMenuItem_Click(object sender,
EventArgs e)
430         {
431             this.WindowState = FormWindowState.Normal;
432         }
433
434         private void TrayBalloonTip(string message, ToolTipIcon
toolTipIcon, int time = 500)
435         {
436             if (minimizeToTrayToolStripMenuItem.Checked)
437             {
438                 notifyIcon1.BalloonTipIcon = toolTipIcon;
439                 notifyIcon1.BalloonTipText = message;
```

```
440                 notifyIcon1.ShowBalloonTip(time);
441             }
442         }
443         #endregion
444
445         private void close_Click(object sender, EventArgs e)
446         {
447             this.Close();
448         }
449
450         private void FormMain_FormClosing(object sender,
FormClosingEventArgs e)
451         {
452             DialogResult reply = MessageBox.Show(this, "Are you
sure you want to close the server?", "Close?",
MessageBoxButtons.YesNo, MessageBoxIcon.Question);
453             if (reply == DialogResult.No)
454                 e.Cancel = true;
455             else
456             {
457                 // Add closing of sockets and stuff here!
458                 _server.Stop();
459                 DatabaseCommunication.Disconnect();
460             }
461         }
462     }
463
464     static class Global
465     {
466         public static FormMain MainWindow { set; get; }
467     }
468 }
```

DatabaseCommunication.cs

```
001 using System;
002 using System.Collections.Generic;
003 using System.Linq;
004 using System.Text;
005 using System.Data;
006 using MySql.Data.MySqlClient; //
http://dev.mysql.com/downloads/connector/net/
007 using System.Net;
008 using System.Threading;
009 using System.Globalization;
010
011 namespace ChatTwo_Server
012 {
013     static class DatabaseCommunication
014     {
015         private static bool _online;
016         public static bool Active
017         {
018             get { return _online; }
019         }
020
021         private static int _version = 0;
022         public static int Version
```

```
023         {
024             get { return _version; }
025         }
026 
027         // StatusIntervalUpdate thread.
028         private static Thread _threadStatusIntervalUpdate;
029 
030         // May have to implement this later. But only needed for
the DateTime objects.
031         //private  static CultureInfo _ci =
CultureInfo.CreateSpecificCulture("en-US");
032 
033         // SqlConnection object is saved here for continued use.
034         private static MySqlConnection _conn;
035 
036         // My attempt at making the MySqlConnection not close
before all tasks are done using it.
037         private static int _SqlWorker = 0;
038 
039         /// <summary>
040         /// Creates the SqlConnection object and starts threaded
methods.
041         /// </summary>
042         /// <param name="user">MySQL "User id" with either root
access or access to the `ChatTwo` database.</param>
043         /// <param name="password">Password of the user.</param>
044         /// <param name="ip">IP address of the machine hosting the
MySQL server.</param>
045         /// <param name="port">Port number the MySQL server is
running on.</param>
046         public static void Connect(string user, string password,
string ip, int port)
047         {
048             MySqlConnectionStringBuilder connBuilder = new
MySqlConnectionStringBuilder();
049             connBuilder.Add("User id", user);
050             connBuilder.Add("Password", password);
051             connBuilder.Add("Network Address", ip);
052             connBuilder.Add("Port", port);
053             connBuilder.Add("Database", "ChatTwo");
054             connBuilder.Add("Connection timeout", 5);
055 
056             // Create the SqlConnection object using the saved IP
address from settings.
057             _conn = new
MySqlConnection(connBuilder.ConnectionString);
058 
059             // Set the status of the database connection to on.
060             _online = true;
061 
062             // Start the thread.
063             _threadStatusIntervalUpdate = new Thread(() =>
StatusIntervalUpdate(connBuilder.ConnectionString));
064             _threadStatusIntervalUpdate.Name =
"StatusIntervalUpdate Thread (StatusIntervalUpdate method)";
065             _threadStatusIntervalUpdate.Start();
066         }
067 
```

```
068         /// <summary>
069         /// Shuts down the database connection and gracefully
stops threaded methods.
070         /// </summary>
071         public static void Disconnect()
072         {
073             // Set the status of the database connection to off.
074             // This also makes the threaded method stop
gracefully.
075             _online = false;
076 
077             // Waits for the thread to end gracefully.
078             if (_threadStatusIntervalUpdate != null)
079                 _threadStatusIntervalUpdate.Join();
080 
081             // Delete the SqlConnection object.
082             _conn = null;
083         }
084 
085         /// <summary>
086         /// Opens the MySqlConnection if it's not already open.
087         /// </summary>
088         private static void Open()
089         {
090             if (_SqlWorker == 0)
091                 _conn.Open();
092             _SqlWorker++;
093         }
094 
095         /// <summary>
096         /// Closes the MySqlConnection if all tasks are down using
it.
097         /// </summary>
098         private static void Close()
099         {
100             _SqlWorker--;
101             if (_SqlWorker == 0 && _conn.State ==
ConnectionState.Open)
102                 _conn.Close();
103         }
104 
105         #region Testing
106         public enum ConnectionTestResult
107         {
108             UnknownError,
109             NoConnection,
110             FailLogin,
111             NoPermission,
112             MissingDatabase,
113             MissingTable,
114             OutDated,
115             Successful
116         }
117 
118         /// <summary>
119         /// Tests the connection to the SQL server and returns a
ConnectionTestResult enum result.
120         /// </summary>
```

```
121         /// <param name="user">MySQL "User id" with either root
access or access to the `ChatTwo` database.</param>
122         /// <param name="password">Password of the user.</param>
123         /// <param name="ip">IP address of the machine hosting the
MySQL server.</param>
124         /// <param name="port">Port number the MySQL server is
running on.</param>
125         public static ConnectionTestResult TestConnection(string
user, string password, string ip, int port)
126         {
127             // Shorter timeout will make the user not have to wait
as long.
128             // (Does not seem to have much of an effect on the
connection timeout.)
129             const int timeout = 5;
130 
131             MySqlConnectionStringBuilder connBuilder = new
MySqlConnectionStringBuilder();
132             connBuilder.Add("User id", user);
133             connBuilder.Add("Password", password);
134             connBuilder.Add("Network Address", ip);
135             connBuilder.Add("Port", port);
136             //connBuilder.Add("Database", "ChatTwo");
137             connBuilder.Add("Connection timeout", timeout);
138 
139             // Test1: Test connection to the server using the IP
address from the settings. Add "Connection Timeout" (even though it
seem not to work).
140             using (MySqlConnection testConn = new
MySqlConnection(connBuilder.ConnectionString))
141             {
142                 // Test2: Test access to the database.
143                 MySqlCommand test2 = new MySqlCommand("USE
`ChatTwo`;", testConn);
144                 test2.CommandTimeout = timeout;
145 
146                 // Test3: Test access to the `Contacts` table and
the `Users` table..
147                 MySqlCommand test3 = new MySqlCommand("SELECT *
FROM `ChatTwo`.`Contacts` WHERE 0 = 1;SELECT * FROM `ChatTwo`.`Users`
WHERE 0 = 1;", testConn);
148                 test3.CommandTimeout = timeout;
149 
150                 // Test4: Test access to the `ServerStatus` table
and get the version number.
151                 MySqlCommand test4 = new MySqlCommand("SELECT
`Version` FROM `ChatTwo`.`ServerStatus`;", testConn);
152                 test4.CommandTimeout = timeout;
153                 int version = -1;
154 
155                 // Run all tests.
156                 try
157                 {
158                     testConn.Open();
159                     test2.ExecuteNonQuery();
160                     test3.ExecuteNonQuery();
161                     MySqlDataReader reader =
test4.ExecuteReader();
```

```
162                     if (reader.Read())
163                     {
164                         version = (int)reader["Version"];
165                     }
166                 }
167                 catch (MySqlException ex)
168                 {
169                     // If one of the tests fail, return an error
message.
170                     switch (ex.Number)
171                     { //
http://dev.mysql.com/doc/refman/5.6/en/error-messages-server.html
172                         case 0:
173                             if (ex.Message.Contains("Access
denied"))
174                             {
175                                 // Login failed
176                                 return
ConnectionTestResult.FailLogin;
177                             }
178                             else
179                             {
180                                 // SQL query timed out
181                                 return
ConnectionTestResult.NoConnection;
182                             }
183                         case 1042:
184                             // (ER_BAD_HOST_ERROR) Message: Can't
get hostname for your address
185                             return
ConnectionTestResult.NoConnection;
186                         case 1044:
187                             // (ER_DBACCESS_DENIED_ERROR) Message:
Access denied for user '%s'@'%s' to database '%s'
188                             return
ConnectionTestResult.NoPermission;
189                         case 1049:
190                             // (ER_BAD_DB_ERROR) Message: Unknown
database '%s'
191                             return
ConnectionTestResult.MissingDatabase;
192                         case 1146:
193                             // (ER_NO_SUCH_TABLE) Message: Table
'%s.%s' doesn't exist
194                             return
ConnectionTestResult.MissingTable;
195                         default:
196                             // Unknown SQL error
197 #if !DEBUG
198                             return
ConnectionTestResult.UnknownError;
199 #else
200                             throw;
201 #endif
202                     }
203                 }
204                 finally
205                 {
```

```
206                     if (testConn.State != ConnectionState.Closed)
207                         testConn.Close();
208                 }
209 
210                 // If the version is old, suggest an update.
211                 if (version < _version)
212                     return ConnectionTestResult.OutDated;
213             }
214 
215             // If nothing bad happens, tell the user the program
is ready.
216             return ConnectionTestResult.Successful;
217         }
218         #endregion
219 
220         #region Creating and updating database
221         /// <summary>
222         /// Create the whole database from scratch.
223         /// </summary>
224         /// <param name="user">MySQL "User id" with either root
access or access to the `ChatTwo` database.</param>
225         /// <param name="password">Password of the user.</param>
226         /// <param name="ip">IP address of the machine hosting the
MySQL server.</param>
227         /// <param name="port">Port number the MySQL server is
running on.</param>
228         public static bool CreateDatabase(string user, string
password, string ip, int port)
229         {
230             int cmdResult = 0;
231 
232             MySqlConnectionStringBuilder connBuilder = new
MySqlConnectionStringBuilder();
233             connBuilder.Add("User id", user);
234             connBuilder.Add("Password", password);
235             connBuilder.Add("Network Address", ip);
236             connBuilder.Add("Port", port);
237             //connBuilder.Add("Database", "ChatTwo");
238             //connBuilder.Add("Connection timeout", 5);
239 
240             using (MySqlConnection tempConn = new
MySqlConnection(connBuilder.ConnectionString))
241             {
242                 using (MySqlCommand cmd = new MySqlCommand(
243                     "CREATE DATABASE IF NOT EXISTS `ChatTwo`;" +
Environment.NewLine +
244                     "USE `ChatTwo`;" + Environment.NewLine +
245                     "" + Environment.NewLine +
246                     "CREATE TABLE IF NOT EXISTS `ServerStatus` ("
+ Environment.NewLine +
247                     "    `Version` INT NOT NULL," +
Environment.NewLine +
248                     "    `CreationDate` TIMESTAMP NOT NULL DEFAULT
CURRENT_TIMESTAMP," + Environment.NewLine +
249                     "    `LastUpdated` TIMESTAMP NOT NULL DEFAULT
CURRENT_TIMESTAMP" + Environment.NewLine +
250                     "    );" + Environment.NewLine +
```

```
251                     "INSERT INTO `ServerStatus` (`Version`)
VALUES(0);" + Environment.NewLine +
252                     "" + Environment.NewLine +
253                     "CREATE TABLE IF NOT EXISTS `Users` (" +
Environment.NewLine +
254                     "    `ID` INT NOT NULL PRIMARY KEY
AUTO_INCREMENT," + Environment.NewLine +
255                     "    `Name` VARCHAR(30) NOT NULL UNIQUE," +
Environment.NewLine +
256                     "    `Password` VARCHAR(28) NOT NULL," +
Environment.NewLine +
257                     "    `Online` TINYINT(1) NOT NULL DEFAULT 0,"
+ Environment.NewLine +
258                     "    `Socket` VARCHAR(51) NULL," +
Environment.NewLine +
259                     "    `LastOnline` TIMESTAMP NOT NULL DEFAULT
CURRENT_TIMESTAMP," + Environment.NewLine +
260                     "    `Registered` TIMESTAMP NOT NULL DEFAULT
CURRENT_TIMESTAMP" + Environment.NewLine +
261                     "    );" + Environment.NewLine +
262                     "CREATE TABLE IF NOT EXISTS `Contacts` (" +
Environment.NewLine +
263                     "    `ID_1` INT NOT NULL," +
Environment.NewLine +
264                     "    `ID_2` INT NOT NULL," +
Environment.NewLine +
265                     "    `1To2` TINYINT(1) NOT NULL DEFAULT 0," +
Environment.NewLine +
266                     "    `2To1` TINYINT(1) NOT NULL DEFAULT 0" +
Environment.NewLine +
267                     "    );" + Environment.NewLine +
268                     "" + Environment.NewLine +
269                     "DROP TRIGGER IF EXISTS `trig_UserInsert`;" +
Environment.NewLine +
270                     "DROP TRIGGER IF EXISTS `trig_UserDeleted`;" +
Environment.NewLine +
271                     "DROP PROCEDURE IF EXISTS `StatusUpdate`;" +
Environment.NewLine +
272                     "DROP PROCEDURE IF EXISTS
`StatusIntervalUpdate`;" + Environment.NewLine +
273                     "DROP PROCEDURE IF EXISTS `ContactsMutual`;" +
Environment.NewLine +
274                     "DROP PROCEDURE IF EXISTS `ContactsALL`;" +
Environment.NewLine +
275                     "DROP PROCEDURE IF EXISTS `ContactsAdd`;" +
Environment.NewLine +
276                     "DROP PROCEDURE IF EXISTS `ContactsRemove`;" +
Environment.NewLine +
277                     "" + Environment.NewLine +
278                     "CREATE TRIGGER `trig_UserInsert`" +
Environment.NewLine +
279                     "    BEFORE INSERT ON `users`" +
Environment.NewLine +
280                     "    FOR EACH ROW" + Environment.NewLine +
281                     "BEGIN" + Environment.NewLine +
282                     "    DELETE FROM `Contacts`" +
Environment.NewLine +
```

```
283                     "        WHERE `ID_1` = NEW.ID" +
Environment.NewLine +
284                     "           OR `ID_2` = NEW.ID;" +
Environment.NewLine +
285                     "END;" + Environment.NewLine +
286                     "" + Environment.NewLine +
287                     "CREATE TRIGGER `trig_UserDeleted`" +
Environment.NewLine +
288                     "    BEFORE DELETE ON `users`" +
Environment.NewLine +
289                     "    FOR EACH ROW" + Environment.NewLine +
290                     "BEGIN" + Environment.NewLine +
291                     "    DELETE FROM `Contacts`" +
Environment.NewLine +
292                     "        WHERE `ID_1` = OLD.ID" +
Environment.NewLine +
293                     "           OR `ID_2` = OLD.ID;" +
Environment.NewLine +
294                     "END;" + Environment.NewLine +
295                     "" + Environment.NewLine +
296                     "CREATE DEFINER=CURRENT_USER PROCEDURE
`StatusUpdate`(" + Environment.NewLine +
297                     "    IN p_ID INT," + Environment.NewLine +
298                     "    IN p_Socket VARCHAR(51)" +
Environment.NewLine +
299                     ")" + Environment.NewLine +
300                     "    MODIFIES SQL DATA" + Environment.NewLine
+
301                     "BEGIN" + Environment.NewLine +
302                     "    UPDATE `Users`" + Environment.NewLine +
303                     "        SET `Online` = 1," +
Environment.NewLine +
304                     "            `Socket` = p_Socket," +
Environment.NewLine +
305                     "            `LastOnline` = CURRENT_TIMESTAMP"
+ Environment.NewLine +
306                     "        WHERE `ID` = p_ID;" +
Environment.NewLine +
307                     "END;" + Environment.NewLine +
308                     "" + Environment.NewLine +
309                     "CREATE DEFINER=CURRENT_USER PROCEDURE
`StatusIntervalUpdate`()" + Environment.NewLine +
310                     "    MODIFIES SQL DATA" + Environment.NewLine
+
311                     "BEGIN" + Environment.NewLine +
312                     "    SELECT `ID` FROM `Users`" +
Environment.NewLine +
313                     "        WHERE (`Online` = 1) " +
Environment.NewLine +
314                     "          AND NOT (`LastOnline` BETWEEN
timestamp(DATE_SUB(NOW(), INTERVAL 10 SECOND)) AND NOW());" +
Environment.NewLine +
315                     "    UPDATE `Users`" + Environment.NewLine +
316                     "        SET `Online` = 0," +
Environment.NewLine +
317                     "            `Socket` = NULL" +
Environment.NewLine +
```

```
318                     "        WHERE (`Online` = 1)" +
Environment.NewLine +
319                     "          AND NOT (`LastOnline` BETWEEN
timestamp(DATE_SUB(NOW(), INTERVAL 10 SECOND)) AND NOW());" +
Environment.NewLine +
320                     "END;" + Environment.NewLine +
321                     "" + Environment.NewLine +
322                     "CREATE DEFINER=CURRENT_USER PROCEDURE
`ContactsMutual`(" + Environment.NewLine +
323                     "    IN p_ID INT" + Environment.NewLine +
324                     ")" + Environment.NewLine +
325                     "    READS SQL DATA" + Environment.NewLine +
326                     "BEGIN" + Environment.NewLine +
327                     "    SELECT IF((`ID_1` = p_ID), `ID_2`,
`ID_1`) AS `ContactID`" + Environment.NewLine +
328                     "        FROM `Contacts`" +
Environment.NewLine +
329                     "        WHERE (`ID_1` = p_ID OR `ID_2` =
p_ID)" + Environment.NewLine +
330                     "          AND `1To2` = 1 AND `2To1` = 1;" +
Environment.NewLine +
331                     "END;" + Environment.NewLine +
332                     "" + Environment.NewLine +
333                     "CREATE DEFINER=CURRENT_USER PROCEDURE
`ContactsAll`(" + Environment.NewLine +
334                     "    IN `p_ID` INT" + Environment.NewLine +
335                     ")" + Environment.NewLine +
336                     "    READS SQL DATA" + Environment.NewLine +
337                     "BEGIN" + Environment.NewLine +
338                     "    SELECT IF((`ID_1` = p_ID), `ID_2`,
`ID_1`) AS ContactID," + Environment.NewLine +
339                     "           IF((`ID_1` = p_ID AND `1To2` = 1)
OR (`ID_2` = p_ID AND `2To1` = 1), 1, 0) AS FromMe," +
Environment.NewLine +
340                     "           IF((`ID_1` = p_ID AND `2To1` = 1)
OR (`ID_2` = p_ID AND `1To2` = 1), 1, 0) AS ToMe" +
Environment.NewLine +
341                     "        FROM `Contacts`" +
Environment.NewLine +
342                     "        WHERE `ID_1` = p_ID OR `ID_2` =
p_ID;" + Environment.NewLine +
343                     "END;" + Environment.NewLine +
344                     "" + Environment.NewLine +
345                     "CREATE DEFINER=CURRENT_USER PROCEDURE
`ContactsAdd`(" + Environment.NewLine +
346                     "    IN `p_ID` INT," + Environment.NewLine +
347                     "    IN `p_ContactID` INT" +
Environment.NewLine +
348                     ")" + Environment.NewLine +
349                     "    MODIFIES SQL DATA" + Environment.NewLine
+
350                     "BEGIN" + Environment.NewLine +
351                     "    IF (SELECT EXISTS(SELECT 1 FROM
`Contacts`" + Environment.NewLine +
352                     "        WHERE (`ID_1` = p_ID AND `ID_2` =
p_ContactID)" + Environment.NewLine +
353                     "          OR (`ID_2` = p_ID AND `ID_1` =
p_ContactID) LIMIT 1) as contactFound) = 1" + Environment.NewLine +
```

```
354                     "    THEN" + Environment.NewLine +
355                     "        UPDATE `contacts`" +
Environment.NewLine +
356                     "        SET `2To1` = IF(`ID_2` = p_ID, 1,
`2To1`), `1To2` = IF(`ID_1` = p_ID, 1, `1To2`)" + Environment.NewLine
+
357                     "        WHERE (`ID_1` = p_ID AND `ID_2` =
p_ContactID)" + Environment.NewLine +
358                     "           OR (`ID_2` = p_ID AND `ID_1` =
p_ContactID);" + Environment.NewLine +
359                     "    ELSE" + Environment.NewLine +
360                     "        INSERT INTO `contacts`(`ID_1`,
`ID_2`, `1To2`, `2To1`)" + Environment.NewLine +
361                     "        VALUES (p_ID, p_ContactID, 1, 0);" +
Environment.NewLine +
362                     "    END IF;" + Environment.NewLine +
363                     "END;" + Environment.NewLine +
364                     "" + Environment.NewLine +
365                     "CREATE DEFINER=CURRENT_USER PROCEDURE
`ContactsRemove`(" + Environment.NewLine +
366                     "    IN `p_ID` INT," + Environment.NewLine +
367                     "    IN `p_ContactID` INT" +
Environment.NewLine +
368                     ")" + Environment.NewLine +
369                     "    MODIFIES SQL DATA" + Environment.NewLine
+
370                     "BEGIN" + Environment.NewLine +
371                     "    UPDATE `contacts`" + Environment.NewLine
+
372                     "    SET `2To1` = IF(`ID_2` = p_ID, 0,
`2To1`), `1To2` = IF(`ID_1` = p_ID, 0, `1To2`)" + Environment.NewLine
+
373                     "    WHERE (`ID_1` = p_ID AND `ID_2` =
p_ContactID)" + Environment.NewLine +
374                     "       OR (`ID_2` = p_ID AND `ID_1` =
p_ContactID);" + Environment.NewLine +
375                     "    DELETE FROM `contacts`" +
Environment.NewLine +
376                     "    WHERE `1To2` = 0 AND `2To1` = 0;" +
Environment.NewLine +
377                     "END;"
378                     , tempConn))
379                 {
380                     try
381                     {
382                         tempConn.Open();
383                         // Execute SQL command.
384                         cmdResult = cmd.ExecuteNonQuery();
385                     }
386                     finally
387                     {
388                         if (tempConn.State !=
System.Data.ConnectionState.Closed)
389                             tempConn.Close();
390                     }
391                 }
392             }
```

```
393             return (cmdResult != 0); // cmdResult content the
number of affected rows.
394         }
395
396         /// <summary>
397         /// Update an out of date database.
398         /// </summary>
399         /// <param name="user">MySQL "User id" with either root
access or access to the `ChatTwo` database.</param>
400         /// <param name="password">Password of the user.</param>
401         /// <param name="ip">IP address of the machine hosting the
MySQL server.</param>
402         /// <param name="port">Port number the MySQL server is
running on.</param>
403         public static bool UpdateDatabase(string user, string
password, string ip, int port)
404         {
405             int cmdResult = 0;
406
407             MySqlConnectionStringBuilder connBuilder = new
MySqlConnectionStringBuilder();
408             connBuilder.Add("User id", user);
409             connBuilder.Add("Password", password);
410             connBuilder.Add("Network Address", ip);
411             connBuilder.Add("Port", port);
412             //connBuilder.Add("Database", "ChatTwo");
413             //connBuilder.Add("Connection timeout", 5);
414
415             using (MySqlConnection tempConn = new
MySqlConnection(connBuilder.ConnectionString))
416             {
417                 // Get the database version number from the
`ServerStatus` table.
418                 int version = -1;
419                 using (MySqlCommand cmd = new MySqlCommand("SELECT
`ChatTwo`.`Version` FROM `ServerStatus`;", tempConn))
420                 {
421                     try
422                     {
423                         Open();
424                         // Execute SQL command.
425                         MySqlDataReader reader =
cmd.ExecuteReader();
426                         if (reader.Read())
427                         {
428                             version = (int)reader["Version"];
429                         }
430                     }
431                     finally
432                     {
433                         Close();
434                     }
435                 }
436
437                 switch (version)
438                 {
439                     case 0:
```

```
440                         throw new NotImplementedException("There
is no update from version 0 (yet).");
441
442                         //using (MySqlCommand cmd = new
MySqlCommand("UPDATE `ServerStatus` SET `Version` = 1, `LastUpdated` =
NOW();", _conn))
443                         //{
444                         //    try
445                         //    {
446                         //        Open();
447                         //        // Execute SQL command.
448                         //        cmdResult =
cmd.ExecuteNonQuery();
449                         //    }
450                         //    finally
451                         //    {
452                         //        Close();
453                         //    }
454                         //}
455                         //break;
456                     default:
457                         break;
458                 }
459             }
460             return (cmdResult != 0); // cmdResult content the
number of affected rows.
461         }
462         #endregion
463
464         #region Common routin
465         /// <summary>
466         /// Create a user on the `Users` table. Username is a
unique column and this medthod will return false if it is already in
use.
467         /// </summary>
468         /// <param name="username">Username of the user to
create.</param>
469         /// <param name="password">28 base64 character hash string
of the user's password.</param>
470         public static bool CreateUser(string username, string
password)
471         {
472             int cmdResult = 0;
473             using (MySqlCommand cmd = new MySqlCommand("INSERT
INTO `Users` (`Name`, `Password`) VALUES(@username, @password);",
_conn))
474             {
475                 // Add parameterized parameters to prevent SQL
injection.
476                 cmd.Parameters.AddWithValue("@username",
username);
477                 cmd.Parameters.AddWithValue("@password",
password);
478
479                 try
480                 {
481                     Open();
482                     // Execute SQL command.
```

```
483                     cmdResult = cmd.ExecuteNonQuery();
484                 }
485                 catch (MySqlException ex)
486                 {
487                     if (ex.Number == 1062) //
http://dev.mysql.com/doc/refman/5.6/en/error-messages-server.html
488                         // (ER_DUP_ENTRY) Message: Duplicate entry
'%s' for key %d
489                         // If the username is already in use.
490                         return false;
491                     throw ex;
492                 }
493                 finally
494                 {
495                     Close();
496                 }
497             }
498             return (cmdResult != 0); // cmdResult content the
number of affected rows.
499         }
500
501         /// <summary>
502         /// Read all database information about a user. Returns an
UserObj with all the informatioin.
503         /// </summary>
504         /// <param name="id">ID number of the requested
user.</param>
505         static public UserObj ReadUser(int id)
506         {
507             UserObj cmdResult = null;
508             using (MySqlCommand cmd = new MySqlCommand("SELECT *
FROM `Users` WHERE `ID` = @id;", _conn))
509             {
510                 // Add parameterized parameters to prevent SQL
injection.
511                 cmd.Parameters.AddWithValue("@id", id);
512
513                 try
514                 {
515                     Open();
516                     // Execute SQL command.
517                     MySqlDataReader reader = cmd.ExecuteReader();
518                     while (reader.Read())
519                     {
520                         cmdResult = new UserObj();
521                         cmdResult.ID = (int)reader["ID"];
522                         cmdResult.Name = (string)reader["Name"];
523                         cmdResult.Online = (bool)reader["Online"];
524
cmdResult.StringSocket(reader["Socket"].ToString());
525                         cmdResult.LastOnline =
(DateTime)reader["LastOnline"];//, _ci);
526                         cmdResult.Registered =
(DateTime)reader["Registered"];//, _ci);
527                     }
528                 }
529                 finally
530                 {
```

```
531                     Close();
532                 }
533             }
534             return cmdResult;
535         }
536 
537         /// <summary>
538         /// Returns an UserObj containing the username and the
userId. Returns null if username/password is incorrect.
539         /// </summary>
540         /// <param name="name">User's username.</param>
541         /// <param name="password">Base64 hash string of the
password.</param>
542         static public UserObj LoginUser(string name, string
password)
543         {
544             UserObj cmdResult = null;
545             using (MySqlCommand cmd = new MySqlCommand("SELECT
`ID`, `Name` FROM `Users` WHERE `Name` = @name AND `Password` =
@password;", _conn))
546             {
547                 // Add parameterized parameters to prevent SQL
injection.
548                 cmd.Parameters.AddWithValue("@name", name);
549                 cmd.Parameters.AddWithValue("@password",
password);
550 
551                 try
552                 {
553                     Open();
554                     // Execute SQL command.
555                     MySqlDataReader reader = cmd.ExecuteReader();
556                     while (reader.Read())
557                     {
558                         cmdResult = new UserObj();
559                         cmdResult.ID = (int)reader["ID"];
560                         cmdResult.Name =
reader["Name"].ToString();
561                     }
562                 }
563                 finally
564                 {
565                     Close();
566                 }
567             }
568             return cmdResult;
569         }
570 
571         /// <summary>
572         /// Updates a user's online status and socket in the
database. Using the StatusUpdate stored procedure.
573         /// </summary>
574         /// <param name="id">User's ID.</param>
575         /// <param name="socket">The IPEndPoint to be written to
the database.</param>
576         static public bool UpdateUser(int id, IPEndPoint socket)
577         {
578             int cmdResult = 0;
```

```
579             using (MySqlCommand cmd = new
MySqlCommand("StatusUpdate", _conn))
580             {
581                 //Set up cmd to reference stored procedure
'StatusUpdate'.
582                 cmd.CommandType =
System.Data.CommandType.StoredProcedure;
583
584                 //Create input parameter (p_ID) and assign a value
(id)
585                 MySqlParameter idParam = new
MySqlParameter("@p_ID", id);
586                 idParam.Direction =
System.Data.ParameterDirection.Input;
587                 cmd.Parameters.Add(idParam);
588                 //Create input parameter (p_Socket) and assign a
value (socket)
589                 MySqlParameter socketParam = new
MySqlParameter("@p_Socket", socket.ToString());
590                 socketParam.Direction =
System.Data.ParameterDirection.Input;
591                 cmd.Parameters.Add(socketParam);
592
593                 try
594                 {
595                     Open();
596                     // Execute SQL command.
597                     cmdResult = cmd.ExecuteNonQuery();
598                 }
599                 finally
600                 {
601                     Close();
602                 }
603             }
604             return (cmdResult != 0); // cmdResult content the
number of affected rows.
605         }
606
607         /// <summary>
608         /// Checks for all the users that haven't reported in for
more than 10 seconds. Using the StatusIntervalUpdate stored procedure.
609         /// </summary>
610         /// <param name="connString">The connection string is
needed here because a separate MySqlConnetion is started for
this.</param>
611         static public void StatusIntervalUpdate(string connString)
// Threaded looping method.
612         {
613             using (MySqlConnection intervalConn = new
MySqlConnection(connString))
614             {
615                 MySqlCommand cmd = new
MySqlCommand("StatusIntervalUpdate", intervalConn);
616                 //Set up cmd to reference stored procedure
'StatusIntervalUpdate'.
617                 cmd.CommandType =
System.Data.CommandType.StoredProcedure;
618
```

```
619                 try
620                 {
621                     while (_online)
622                     {
623                         intervalConn.Open();
624                         // Execute SQL command.
625                         MySqlDataReader reader =
cmd.ExecuteReader();
626                         while (reader.Read())
627                         {
628                             int userId = (int)reader["ID"];
629                             Thread userStatusChange = new
Thread(() => UserStatusChanged(userId, false));
630                             userStatusChange.Name = "UserChange
Thread (UserStatusChanged method)";
631                             userStatusChange.Start();
632                         }
633                         intervalConn.Close();
634                         Thread.Sleep(1000); // 1 seconds.
635                     }
636                 }
637                 catch (Exception ex)
638                 {
639                     System.Diagnostics.Debug.WriteLine("### " +
_threadStatusIntervalUpdate.Name + " has crashed:");
640                     System.Diagnostics.Debug.WriteLine("### " +
ex.Message);
641                     System.Diagnostics.Debug.WriteLine("### " +
ex.ToString());
642                 }
643                 finally
644                 {
645                     if (intervalConn.State ==
ConnectionState.Open)
646                         intervalConn.Close();
647                 }
648             }
649         }
650
651         /// <summary>
652         /// When a user changes status. For example coming online
or going offline.
653         /// This will find all online mutual contacts and fire an
OnUserStatusChange for each.
654         /// </summary>
655         /// <param name="userId">ID number of the user changing
status.</param>
656         /// <param name="comesOnline">Set true if they are coming
online, or false if they are going offline.</param>
657         private static void UserStatusChanged(int userId, bool
comesOnline) // Threaded method.
658         {
659             // Should make the ContactsMutual stored procedure
only return contacts that are online.
660             // Would make this method faster by only having it
make one query. Just not sure how.
661             List<int> contactIds = new List<int>();
```

```
662             using (MySqlCommand cmd = new
MySqlCommand("ContactsMutual", _conn))
663             {
664                 //Set up cmd to reference stored procedure
'ContactsMutual'.
665                 cmd.CommandType =
System.Data.CommandType.StoredProcedure;
666
667                 //Create input parameter (p_ID) and assign a value
(id)
668                 MySqlParameter idParam = new
MySqlParameter("@p_ID", userId);
669                 idParam.Direction =
System.Data.ParameterDirection.Input;
670                 cmd.Parameters.Add(idParam);
671
672                 try
673                 {
674                     Open();
675                     // Execute SQL command.
676                     MySqlDataReader reader = cmd.ExecuteReader();
677                     while (reader.Read())
678                     {
679                         contactIds.Add((int)reader["ContactID"]);
680                     }
681                 }
682                 finally
683                 {
684                     Close();
685                 }
686             }
687             if (contactIds.Count != 0)
688             {
689                 // This is all created in the method, so no chance
of SQL injection.
690                 string users = String.Join(" OR `ID` = ",
contactIds);
691                 contactIds.Clear();
692                 using (MySqlCommand cmd = new MySqlCommand("SELECT
`ID` FROM `Users` WHERE `Online` = 1 AND (`ID` = " + users + ");",
_conn))
693                 {
694
695                     try
696                     {
697                         Open();
698                         // Execute SQL command.
699                         MySqlDataReader reader =
cmd.ExecuteReader();
700                         while (reader.Read())
701                         {
702                             contactIds.Add((int)reader["ID"]);
703                         }
704                     }
705                     finally
706                     {
707                         Close();
708                     }
```

```
709                 }
710                 // And finally we have a list of all the contacts
that are online and we can message them.
711                 foreach (int contactId in contactIds)
712                 {
713                     // Fire an OnUserStatusChange event.
714                     UserStatusChangeEventArgs args = new
UserStatusChangeEventArgs();
715                     args.TellId = contactId;
716                     args.IdIs = userId;
717                     args.Online = comesOnline;
718                     OnUserStatusChange(args);
719                 }
720             }
721         }
722         #endregion
723
724         private static void
OnUserStatusChange(UserStatusChangeEventArgs e)
725         {
726             EventHandler<UserStatusChangeEventArgs> handler =
UserStatusChange;
727             if (handler != null)
728             {
729                 handler(null, e);
730             }
731         }
732         public static event
EventHandler<UserStatusChangeEventArgs> UserStatusChange;
733     }
734
735     public class UserStatusChangeEventArgs : EventArgs
736     {
737         public int TellId { get; set; }
738         public int IdIs { get; set; }
739         public bool Online { get; set; }
740         public IPEndPoint Socket { get; set; }
741     }
742 }
```

ChatTwo_Server_Protocol.cs

```
001 using System;
002 using System.Collections.Generic;
003 using System.Linq;
004 using System.Text;
005 using System.Net;
006 using System.Net.Sockets;
007
008 namespace ChatTwo_Server
009 {
010     class ChatTwo_Server_Protocol
011     {
012         private static List<UserObj> _users = new List<UserObj>();
013         public static List<UserObj> Users
014         {
015             get { return _users; }
016             set { _users = value; }
```

```
017         }
018
019         public static void MessageReceivedHandler(object sender,
PacketReceivedEventArgs args)
020         {
021             if (!DatabaseCommunication.Active)
022                 throw new NotImplementedException("Database
connection was not active and a reply for this have not been
implemented yet.");
023                 // Need to add a simple debug message here, but
this works as a great breakpoint until then.
024                 // Also need to make some kind of error message I
can send back to the client.
025
026             if (args.Data[0] == 0x92 )
027             {
028                 string sharedSecret;
029                 // Position of the Type byte is 30
(SignatureByteLength + MacByteLength + TimezByteLength +
UserIdByteLength).
030                 ChatTwo_Protocol.MessageType type =
(ChatTwo_Protocol.MessageType)args.Data[ChatTwo_Protocol.SignatureByte
Length + ByteHelper.HashByteLength + 4 + 4];
031                 if (type ==
ChatTwo_Protocol.MessageType.CreateUser)
032                 {
033                     sharedSecret =
ChatTwo_Protocol.DefaultSharedSecret;
034                 }
035                 else if (type ==
ChatTwo_Protocol.MessageType.Login)
036                 {
037 #if DEBUG
038                     byte[] test = ByteHelper.SubArray(args.Data,
ChatTwo_Protocol.SignatureByteLength + ByteHelper.HashByteLength + 4);
039 #endif
040                     // Don't take the Timez as part of the
sharedSecret. This is mostly because of a problem I have in the client
where I make the sharedSecrt before I add the Timez.
041                     sharedSecret =
ByteHelper.GetHashString(ByteHelper.SubArray(args.Data,
ChatTwo_Protocol.SignatureByteLength + ByteHelper.HashByteLength +
4));
042                 }
043                 else
044                 {
045                     // Position of the UserID bytes is 26
(SignatureByteLength + MacByteLength + TimezByteLength) with a length
of 4.
046                     int userId = ByteHelper.ToInt32(args.Data,
ChatTwo_Protocol.SignatureByteLength + ByteHelper.HashByteLength + 4);
047                     sharedSecret = _users.Find(x => x.ID ==
userId).Secret;
048                 }
049
050
051                 if (ChatTwo_Protocol.ValidateMac(args.Data,
sharedSecret))
```

```
052                 {
053                     Message message =
ChatTwo_Protocol.MessageReceivedHandler(args);
054 
055                     switch (message.Type)
056                     {
057                         case
ChatTwo_Protocol.MessageType.CreateUser:
058                             {
059                                 string passwordHash =
Convert.ToBase64String(message.Data, 0, ByteHelper.HashByteLength);
060                                 string username =
Encoding.Unicode.GetString(ByteHelper.SubArray(message.Data,
ByteHelper.HashByteLength));
061                                 bool worked =
DatabaseCommunication.CreateUser(username, passwordHash);
062                                 if (worked)
063                                 {
064                                     // Uesr creation worked!
065                                     MessageToIp(message.Ip,
ChatTwo_Protocol.MessageType.CreateUserReply, new byte[] { 0x00 });
066                                 }
067                                 else
068                                 {
069                                     // Some error prevented the
user from being created. Best guess is that a user with that name
already exist.
070                                     MessageToIp(message.Ip,
ChatTwo_Protocol.MessageType.CreateUserReply, new byte[] { 0x01 });
071                                 }
072                                 break;
073                             }
074                         case ChatTwo_Protocol.MessageType.Login:
075                             {
076                                 string passwordHash =
Convert.ToBase64String(message.Data, 0, ByteHelper.HashByteLength);
077                                 string username =
Encoding.Unicode.GetString(ByteHelper.SubArray(message.Data,
ByteHelper.HashByteLength));
078                                 UserObj user =
DatabaseCommunication.LoginUser(username, passwordHash);
079                                 if (user == null)
080                                 {
081                                     // Have to send back a
LoginReply message here with a "wrong username/password" error.
082                                     MessageToIp(message.Ip,
ChatTwo_Protocol.MessageType.LoginReply, new byte[] { 0x01 });
083                                     return;
084                                 }
085                                 if (_users.Any(x => x.ID ==
user.ID))
086                                 {
087                                     // Have to send back a
LoginReply message here with a "User is already online" error.
088                                     MessageToIp(message.Ip,
ChatTwo_Protocol.MessageType.LoginReply, new byte[] { 0x02 });
089                                     return;
090                                 }
```

```
091                                 user.Secret = sharedSecret;
092                                 user.Socket = message.Ip;
093                                 _users.Add(user);
094                                 MessageToUser(user.ID,
ChatTwo_Protocol.MessageType.LoginReply,
ByteHelper.ConcatinateArray(new byte[] { 0x00 },
BitConverter.GetBytes(user.ID)), user.Name);
095
DatabaseCommunication.UpdateUser(user.ID, user.Socket);
096                                 break;
097                             }
098                         case ChatTwo_Protocol.MessageType.Status:
099                             {
100                                 UserObj user = _users.Find(x =>
x.ID == message.From);
101                                 if (user.Socket != message.Ip)
102                                 {
103                                     // Message all contacts of the
user with the new IP change!!!
104                                     user.Socket = message.Ip;
105                                 }
106
DatabaseCommunication.UpdateUser(user.ID, user.Socket);
107                                 break;
108                             }
109                     }
110                 }
111                 else
112                     throw new NotImplementedException("Could not
validate the MAC of the received message.");
113                     // Need to add a simple debug message here,
but this works as a great breakpoint until then.
114             }
115             else
116                 throw new NotImplementedException("Could not
validate the signature of the received message. The signature was
\"0x" + args.Data[0] + "\" but only \"0x92\" is allowed.");
117                 // Need to add a simple debug message here, but
this works as a great breakpoint until then.
118         }
119
120         public static void MessageToIp(IPEndPoint toIp,
ChatTwo_Protocol.MessageType type, byte[] data = null, string text =
null)
121         {
122             Message message = new Message();
123             message.From = ChatTwo_Protocol.ServerReserrvedUserID;
124             message.Type = type;
125             if (data != null && data.Length != 0)
126                 message.Data = data;
127             if (!String.IsNullOrEmpty(text))
128                 message.Text = text;
129             message.Ip = toIp;
130             MessageTransmissionHandler(message);
131         }
132
```

```
133         public static void MessageToUser(int to,
ChatTwo_Protocol.MessageType type, byte[] data = null, string text =
null)
134         {
135             Message message = new Message();
136             message.From = ChatTwo_Protocol.ServerReserrvedUserID;
137             message.To = to;
138             message.Type = type;
139             if (data != null && data.Length != 0)
140                 message.Data = data;
141             if (!String.IsNullOrEmpty(text))
142                 message.Text = text;
143             if (_users.Any(x => x.ID == to))
144                 message.Ip = _users.Find(x => x.ID == to).Socket;
145             MessageTransmissionHandler(message);
146         }
147
148         public static void MessageTransmissionHandler(Message
message)
149         {
150             string sharedSecret;
151             if (message.Type ==
ChatTwo_Protocol.MessageType.CreateUserReply)
152             {
153                 sharedSecret =
ChatTwo_Protocol.DefaultSharedSecret;
154             }
155             else if (message.Type ==
ChatTwo_Protocol.MessageType.LoginReply && message.To == 0)
156             {
157                 // !? This will only happen if the login attempt
failed.
158                 // But because the login attempt failed, I don't
save a UserObj object in the _users list, which in turn mean I don't
have a sharedSecret saved!
159                 //sharedSecret =
160                 throw new NotImplementedException("Login attempt
failed." + Environment.NewLine + "But because the login attempt
failed, I don't save a UserObj object in the _users list, which in
turn mean I don't have a sharedSecret saved!");
161             }
162             else
163             {
164                 sharedSecret = _users.Find(x => x.ID ==
message.To).Secret;
165             }
166
167             byte[] messageBytes =
ChatTwo_Protocol.MessageTransmissionHandler(message);
168
169             messageBytes =
ChatTwo_Protocol.AddSignatureAndMac(messageBytes, sharedSecret);
170
171             // Fire an OnMessageTransmission event.
172             PacketTransmissionEventArgs args = new
PacketTransmissionEventArgs();
173             args.Destination = message.Ip;
174             args.PacketContent = messageBytes;
```

```
175             OnMessageTransmission(args);
176         }
177 
178         public static void TellUserAboutContactstatusChange(object
sender, UserStatusChangeEventArgs args)
179         {
180             byte[] dataBytes =
ByteHelper.ConcatinateArray(BitConverter.GetBytes(args.IdIs), new
byte[] { Convert.ToByte(args.Online) });
181             if (args.Online)
182             {
183                 IPEndPoint socket;
184                 if (args.Socket != null)
185                     socket = args.Socket;
186                 else if (_users.Any(x => x.ID == args.IdIs))
187                     socket = _users.Find(x => x.ID ==
args.IdIs).Socket;
188                 else
189                     // This shouldn't really happen. I should make
the server simply manage online status and sockets only in the memory,
and not in the database.
190                     throw new NotImplementedException("Could not
find a socket for the user[" + args.IdIs + "].");
191                 // 0x01 for UDP only.
192                 byte[] socketBytes =
ByteHelper.ConcatinateArray(new byte[] {0x01},
BitConverter.GetBytes(args.Socket.Port),
args.Socket.Address.GetAddressBytes());
193                 dataBytes = ByteHelper.ConcatinateArray(dataBytes,
socketBytes);
194             }
195             MessageToUser(args.TellId,
ChatTwo_Protocol.MessageType.ContactStatus, dataBytes);
196         }
197 
198         private static void
OnMessageTransmission(PacketTransmissionEventArgs e)
199         {
200             EventHandler<PacketTransmissionEventArgs> handler =
MessageTransmission;
201             if (handler != null)
202             {
203                 handler(null, e);
204             }
205         }
206         public static event
EventHandler<PacketTransmissionEventArgs> MessageTransmission;
207     }
208 }
```

Client Code

FormMain.cs

```
001 using System;
002 using System.Collections.Generic;
003 using System.ComponentModel;
```

```
004 using System.Data;
005 using System.Drawing;
006 using System.Linq;
007 using System.Text;
008 using System.Windows.Forms;
009
010 namespace ChatTwo
011 {
012     public partial class FormMain : Form
013     {
014         UdpCommunication _client;
015
016         Dictionary<int, FormChat> _chats = new Dictionary<int,
FormChat>();
017
018         public FormMain()
019         {
020             InitializeComponent();
021
022 #if DEBUG
023             this.Text += " (DEBUG)";
024 #endif
025
026             _client = new UdpCommunication();
027             _client.PacketReceived +=
ChatTwo_Client_Protocol.MessageReceivedHandler;
028             ChatTwo_Client_Protocol.MessageTransmission +=
_client.SendPacket;
029
030 #if DEBUG
031             // Localhost as server addressed used for easier
testing.
032             ChatTwo_Client_Protocol.ServerAddress = new
System.Net.IPEndPoint(new System.Net.IPAddress(new byte[] { 127, 0, 0,
1 }), 9020);
033 #else
034             // My server IP and port.
035             // Need to make this changable.
036             ChatTwo_Client_Protocol.ServerAddress = new
System.Net.IPEndPoint(new System.Net.IPAddress(new byte[] { 87, 52,
32, 46 }), 9020);
037 #endif
038             MessageBox.Show(this, "ServerAddress set to " +
ChatTwo_Client_Protocol.ServerAddress.ToString() + ".", "ChatTwo
ServerAddress");
039
040             StartUdpClient(0);
041             //StartUdpClient(9020);
042
043             notifyIcon1.BalloonTipTitle = this.Text;
044             notifyIcon1.Text = this.Text;
045             notifyIcon1.Icon =
Icon.ExtractAssociatedIcon(Application.ExecutablePath);
046             this.Icon =
Icon.ExtractAssociatedIcon(Application.ExecutablePath);
047         }
048
049         private bool StartUdpClient(int port)
```

```
050         {
051             bool worked = _client.Start(port);
052 #if DEBUG
053             if (worked)
054                 MessageBox.Show(this, "UDP server started on port
" + _client.Port + ".", "UdpCommunication");
055             else
056                 MessageBox.Show(this, "UDP server failed on port "
+ port + ".", "UdpCommunication");
057 #endif
058             return worked;
059         }
060 
061         private void loginToolStripMenuItem_Click(object sender,
EventArgs e)
062         {
063             using (FormLogin loggingin = new FormLogin())
064             {
065                 loggingin.ShowDialog(this);
066                 if (loggingin.DialogResult ==
System.Windows.Forms.DialogResult.Yes) // The FormLogin's DialogResult
is only set to "Yes" if it was closed by a successful login.
067                 {
068                     // !?!?!?! Logged in?
069                     MessageBox.Show(this, "You have successfully
logged in. This is about as far as the prototype goes.", "Login
Successful", MessageBoxButtons.OK, MessageBoxIcon.Information);
070                     btnAddContact.Enabled = true;
071                     dgvContacts.Enabled = true;
072                     loginToolStripMenuItem.Enabled = false;
073                     logoutToolStripMenuItem.Enabled = true;
074                     toolStripStatusLabel1.Text = "Logged in as " +
loggingin.Username;
075                 }
076             }
077         }
078 
079         private void logoutToolStripMenuItem_Click(object sender,
EventArgs e)
080         {
081             MessageBox.Show(this, "This feature is sadly not
implemented yet." + Environment.NewLine +
082                 "" + Environment.NewLine +
083                 "Currently the server just detects that you have
timed out, but it doesn't forget you were online." +
Environment.NewLine +
084                 "To try again, please restart the server.",
"Logout", MessageBoxButtons.OK, MessageBoxIcon.Information);
085             btnAddContact.Enabled = false;
086             dgvContacts.Enabled = false;
087             logoutToolStripMenuItem.Enabled = false;
088             ChatTwo_Client_Protocol.LogOut();
089             loginToolStripMenuItem.Enabled = true;
090         }
091 
092         private void btnAddContact_Click(object sender, EventArgs
e)
093         {
```

```
094             MessageBox.Show(this, "This feature is sadly not
implemented yet.", "Add Contact", MessageBoxButtons.OK,
MessageBoxIcon.Information);
095         }
096
097         private void closeToolStripMenuItem_Click(object sender,
EventArgs e)
098         {
099             CloseForm();
100         }
101
102         #region Closing Minimize to Tray
103         bool _closing = false;
104         private void CloseForm()
105         {
106             // Exiting the program for real.
107             _closing = true;
108             this.Close();
109         }
110
111         private void FormMain_FormClosing(object sender,
FormClosingEventArgs e)
112         {
113             // Check if we are exiting the program, or just hiding
it.
114             if (!_closing)
115             {
116                 e.Cancel = true;
117                 this.Hide();
118                 TrayBalloonTip("Minimized to tray",
ToolTipIcon.None);
119                 return;
120             }
121
122             // We are exting the program, stop all threaded
workers and stuff.
123             _client.Stop();
124             if (ChatTwo_Client_Protocol.LoggedIn)
125                 ChatTwo_Client_Protocol.LogOut();
126         }
127
128         private void notifyIcon1_MouseDoubleClick(object sender,
MouseEventArgs e)
129         {
130             RestoreForm();
131         }
132
133         private void RestoreToolStripMenuItem_Click(object sender,
EventArgs e)
134         {
135             RestoreForm();
136         }
137
138         private void RestoreForm()
139         {
140             this.Show();
141             this.WindowState = FormWindowState.Normal;
142         }
```

```
143
144         private void TrayBalloonTip(string message, ToolTipIcon
toolTipIcon, int time = 500)
145         {
146             notifyIcon1.BalloonTipIcon = toolTipIcon;
147             notifyIcon1.BalloonTipText = message;
148             notifyIcon1.ShowBalloonTip(time);
149         }
150         #endregion
151     }
152 }
```

FormLogin.cs

```
001 using System;
002 using System.Collections.Generic;
003 using System.ComponentModel;
004 using System.Data;
005 using System.Drawing;
006 using System.Linq;
007 using System.Text;
008 using System.Windows.Forms;
009
010 namespace ChatTwo
011 {
012     public partial class FormLogin : Form
013     {
014         private bool _waitingForLoginReply = false;
015
016         private string _loggedInUsername;
017         public string Username
018         {
019             get { return _loggedInUsername; }
020         }
021
022         public FormLogin()
023         {
024             InitializeComponent();
025
026             lblResult.Text = "";
027             tbxUsername.Focus();
028             // Set the position of the window to the center of the
parent.
029             this.StartPosition = FormStartPosition.CenterParent;
030
031             ChatTwo_Client_Protocol.LoginReply += LoginReply;
032         }
033
034         private void btnRegister_Click(object sender, EventArgs e)
035         {
036             using (FormRegister registering = new FormRegister())
037             {
038                 registering.ShowDialog(this);
039                 if (registering.DialogResult ==
System.Windows.Forms.DialogResult.Yes) // The FormRegister's
DialogResult is only set to "Yes" if it was closed by a successful
user creation.
040                 {
```

```
041                     tbxUsername.Text = registering.Username;
042                     tbxPassword.Focus();
043                 }
044             }
045         }
046
047         private void ResetWindow()
048         {
049             lblUsername.ForeColor = Color.Black;
050             tbxUsername.ForeColor = Color.Black;
051             lblPassword.ForeColor = Color.Black;
052             tbxPassword.ForeColor = Color.Black;
053             lblResult.ForeColor = Color.Black;
054             lblResult.Text = "";
055         }
056
057         private void ResetControls()
058         {
059             btnLogin.Enabled = true;
060             btnRegister.Enabled = true;
061             btnCancel.Enabled = true;
062
063             _waitingForLoginReply = false;
064         }
065
066         private void btnLogin_Click(object sender, EventArgs e)
067         {
068             ResetWindow();
069
070             // If there is no username entered.
071             if (tbxUsername.Text == "")
072             {
073                 lblUsername.ForeColor = Color.Red;
074                 tbxUsername.ForeColor = Color.Red;
075                 lblResult.ForeColor = Color.Red;
076                 lblResult.Text = "You did not enter a username.";
077                 return;
078             }
079
080             // If there is no password entered.
081             if (tbxPassword.Text == "")
082             {
083                 lblPassword.ForeColor = Color.Red;
084                 tbxPassword.ForeColor = Color.Red;
085                 lblResult.ForeColor = Color.Red;
086                 lblResult.Text = "You did not enter a password.";
087                 return;
088             }
089
090             btnLogin.Enabled = false;
091             btnRegister.Enabled = false;
092
093             lblResult.Text = "Contacting server...";
094             _waitingForLoginReply = true;
095             byte[] passwordHash =
ByteHelper.GetHashBytes(Encoding.Unicode.GetBytes(tbxPassword.Text));
```

```
096 ChatTwo_Client_Protocol.MessageToServer(ChatTwo_Protocol.MessageType.Login, passwordHash, tbxUsername.Text);
097             timer1.Start();
098         }
099 
100         public void LoginReply(object sender, LoginReplyEventArgs args)
101         {
102             if (lblResult.InvokeRequired)
103             { // Needed for multi-threading cross calls.
104                 this.Invoke(new Action<object, LoginReplyEventArgs>(this.LoginReply), new object[] { sender, args });
105             }
106             else
107             {
108                 if (_waitingForLoginReply)
109                 {
110                     _waitingForLoginReply = false;
111                     timer1.Stop();
112                     if (args.Success)
113                     {
114                         lblResult.Text = "Login successful!";
115                         _loggedInUsername = args.Name;
116                         this.Close();
117                         this.DialogResult = System.Windows.Forms.DialogResult.Yes;
118                         return;
119                     }
120                     else
121                     {
122                         lblResult.ForeColor = Color.Red;
123                         lblResult.Text = args.Message;
124                         btnRegister.Enabled = true;
125                     }
126                 }
127             }
128         }
129 
130         private void timer1_Tick(object sender, EventArgs e)
131         {
132             if (_waitingForLoginReply)
133             {
134                 ResetControls();
135                 timer1.Stop();
136                 lblResult.ForeColor = Color.Red;
137                 lblResult.Text = "No response from server.";
138             }
139         }
140 
141         private void btnCancel_Click(object sender, EventArgs e)
142         {
143             if (_waitingForLoginReply)
144             {
145                 ResetControls();
146                 timer1.Stop();
147                 ResetWindow();
148                 lblResult.Text = "Login canceled.";
```

```
149             }
150             else
151             {
152                 this.Close();
153                 return;
154             }
155         }
156     }
157 }
```

FormRegister.cs

```
001 using System;
002 using System.Collections.Generic;
003 using System.ComponentModel;
004 using System.Data;
005 using System.Drawing;
006 using System.Linq;
007 using System.Text;
008 using System.Windows.Forms;
009
010 namespace ChatTwo
011 {
012     public partial class FormRegister : Form
013     {
014         public string Username
015         {
016             get { return tbxUsername.Text; }
017         }
018
019         private bool _waitingForCreateUserReply = false;
020
021         private int _newUserId = 0;
022         public int UserId
023         {
024             get { return _newUserId; }
025         }
026
027         public FormRegister()
028         {
029             InitializeComponent();
030
031             lblResult.Text = "";
032             tbxUsername.Focus();
033             // Set the position of the window to the center of the
parent.
034             this.StartPosition = FormStartPosition.CenterParent;
035
036             ChatTwo_Client_Protocol.CreateUserReply +=
CreateUserReply;
037         }
038
039         private void ResetWindow()
040         {
041             lblUsername.ForeColor = Color.Black;
042             tbxUsername.ForeColor = Color.Black;
043             lblPassword1.ForeColor = Color.Black;
044             tbxPassword1.ForeColor = Color.Black;
```

```
045             lblPassword2.ForeColor = Color.Black;
046             tbxPassword2.ForeColor = Color.Black;
047             lblResult.ForeColor = Color.Black;
048             lblResult.Text = "";
049         }
050
051         private void btnRegister_Click(object sender, EventArgs e)
052         {
053             ResetWindow();
054
055             // If there is no username entered.
056             if (tbxUsername.Text == "")
057             {
058                 lblUsername.ForeColor = Color.Red;
059                 tbxUsername.ForeColor = Color.Red;
060                 lblResult.ForeColor = Color.Red;
061                 lblResult.Text = "You did not enter a username.";
062                 return;
063             }
064
065             // If there is no password entered.
066             if (tbxPassword1.Text == "")
067             {
068                 lblPassword1.ForeColor = Color.Red;
069                 tbxPassword1.ForeColor = Color.Red;
070                 lblResult.ForeColor = Color.Red;
071                 lblResult.Text = "You did not enter a password.";
072                 return;
073             }
074
075             // If the password and the confirm password textboxes
aren't the same.
076             if (tbxPassword1.Text != tbxPassword2.Text)
077             {
078                 lblPassword2.ForeColor = Color.Red;
079                 tbxPassword2.ForeColor = Color.Red;
080                 lblResult.ForeColor = Color.Red;
081                 lblResult.Text = "The two passwords are not the
same.";
082                 return;
083             }
084
085             // If the password is too short.
086             // (I hate strict password rules! If it is not a bank
or social security thing, don't force the uesr to make insane
passwords.)
087             if (tbxPassword1.Text.Length < 4)
088             {
089                 lblPassword1.ForeColor = Color.Red;
090                 tbxPassword1.ForeColor = Color.Red;
091                 lblResult.ForeColor = Color.Red;
092                 lblResult.Text = "The password is too short.
Please use 4 or more characters.";
093                 return;
094             }
095
096             btnRegister.Enabled = false;
097             btnCancel.Enabled = false;
```

```
098
099             lblResult.Text = "Contacting server...";
100             _waitingForCreateUserReply = true;
101             byte[] passwordHash =
ByteHelper.GetHashBytes(Encoding.Unicode.GetBytes(tbxPassword1.Text));
102
ChatTwo_Client_Protocol.MessageToServer(ChatTwo_Protocol.MessageType.C
reateUser, passwordHash, tbxUsername.Text);
103             timer1.Start();
104         }
105
106         public void CreateUserReply(object sender,
CreateUserReplyEventArgs args)
107         {
108             if (lblResult.InvokeRequired)
109             { // Needed for multi-threading cross calls.
110                 this.Invoke(new Action<object,
CreateUserReplyEventArgs>(this.CreateUserReply), new object[] {
sender, args });
111             }
112             else
113             {
114                 if (_waitingForCreateUserReply)
115                 {
116                     _waitingForCreateUserReply = false;
117                     timer1.Stop();
118                     if (args.Success)
119                     {
120                         lblResult.Text = "User created
successful!";
121                         //_newUserId = args.ID;
122                         this.Close();
123                         this.DialogResult =
System.Windows.Forms.DialogResult.Yes;
124                         return;
125                     }
126                     else
127                     {
128                         lblResult.ForeColor = Color.Red;
129                         lblResult.Text = args.Message;
130                         btnRegister.Enabled = true;
131                         btnCancel.Enabled = true;
132                     }
133                 }
134             }
135         }
136
137         private void timer1_Tick(object sender, EventArgs e)
138         {
139             if (_waitingForCreateUserReply)
140             {
141                 _waitingForCreateUserReply = false;
142                 btnRegister.Enabled = true;
143                 btnCancel.Enabled = true;
144                 timer1.Stop();
145                 lblResult.ForeColor = Color.Red;
146                 lblResult.Text = "No response from server.";
147             }
```

```
148         }
149
150         private void btnCancel_Click(object sender, EventArgs e)
151         {
152             if (!_waitingForCreateUserReply)
153             {
154                 this.Close();
155                 return;
156             }
157         }
158     }
159 }
```

FormChat.cs

```
01 using System;
02 using System.Collections.Generic;
03 using System.ComponentModel;
04 using System.Data;
05 using System.Drawing;
06 using System.Linq;
07 using System.Text;
08 using System.Windows.Forms;
09
10 namespace ChatTwo
11 {
12     public partial class FormChat : Form
13     {
14         public FormChat()
15         {
16             InitializeComponent();
17         }
18     }
19 }
```

ChatTwo_Client_Protocol.cs

```
001 using System;
002 using System.Collections.Generic;
003 using System.Linq;
004 using System.Text;
005 using System.Net;
006 using System.Net.Sockets;
007 using System.Threading;
008
009 namespace ChatTwo
010 {
011     class ChatTwo_Client_Protocol
012     {
013         private static List<UserObj> _contacts = new
List<UserObj>();
014         public static List<UserObj> Contacts
015         {
016             get { return _contacts; }
017             set { _contacts = value; }
018         }
019
020         private static string _serverSharedSecret = "";
```

```
021         public static string ServerSharedSecret
022         {
023             get { return _serverSharedSecret; }
024             set { _serverSharedSecret = value; }
025         }
026
027         private static IPEndPoint _serverAddress;
028         public static IPEndPoint ServerAddress
029         {
030             get { return _serverAddress; }
031             set { _serverAddress = value; }
032         }
033
034         private static UserObj _user = new UserObj();
035         public static UserObj User
036         {
037             get { return _user; }
038             set { _user = value; }
039         }
040
041         private static bool _loggedIn = false;
042         public static bool LoggedIn
043         {
044             get { return _loggedIn; }
045             set { _loggedIn = value; }
046         }
047
048         private static Thread _threadKeepalive;
049
050         public static void LogIn(int userId)
051         {
052             _user.ID = userId;
053             _loggedIn = true;
054
055             _threadKeepalive = new Thread(() => Keepalive());
056             _threadKeepalive.Name = "Keepalive Thread (Keepalive
method)";
057             _threadKeepalive.Start();
058         }
059
060         public static void LogOut()
061         {
062             _loggedIn = false;
063
064             //_threadKeepalive.Abort();
065             _threadKeepalive.Join();
066         }
067
068         private static void Keepalive() // Threaded looping
method.
069         {
070             try
071             {
072                 while (_loggedIn)
073                 {
074                     Thread.Sleep(500);
```

```
075 ChatTwo_Client_Protocol.MessageToServer(ChatTwo_Protocol.MessageType.Status, null, null);
076                 }
077             }
078             catch (Exception ex)
079             {
080                 System.Diagnostics.Debug.WriteLine("### " + _threadKeepalive.Name + " has crashed:");
081                 System.Diagnostics.Debug.WriteLine("### " + ex.Message);
082                 System.Diagnostics.Debug.WriteLine("### " + ex.ToString());
083             }
084         }
085 
086         public static void MessageReceivedHandler(object sender, PacketReceivedEventArgs args)
087         {
088             if (args.Data[0] == 0x92)
089             {
090                 string sharedSecret;
091                 // Position of the Type byte is 30 (SignatureByteLength + MacByteLength + TimezByteLength + UserIdByteLength).
092                 ChatTwo_Protocol.MessageType type = (ChatTwo_Protocol.MessageType)args.Data[ChatTwo_Protocol.SignatureByteLength + ByteHelper.HashByteLength + 4 + 4];
093                 if (type == ChatTwo_Protocol.MessageType.CreateUserReply)
094                 {
095                     sharedSecret = ChatTwo_Protocol.DefaultSharedSecret;
096                 }
097                 else if (type == ChatTwo_Protocol.MessageType.LoginReply)
098                 {
099                     sharedSecret = ServerSharedSecret;
100                 }
101                 else
102                 {
103                     // Position of the UserID bytes is 26 (SignatureByteLength + MacByteLength + TimezByteLength) with a length of 4.
104                     int userId = ByteHelper.ToInt32(args.Data, ChatTwo_Protocol.SignatureByteLength + ByteHelper.HashByteLength + 4);
105                     sharedSecret = _contacts.Find(x => x.ID == userId).Secret;
106                 }
107 
108                 if (ChatTwo_Protocol.ValidateMac(args.Data, sharedSecret))
109                 {
110                     Message message = ChatTwo_Protocol.MessageReceivedHandler(args);
111 
112                     IPEndPoint messageSender = message.Ip;
113                     //type = message.Type;
```

```
114                     byte[] messageBytes = message.Data;
115
116                     byte[] messageData = new byte[0];
117                     string messageText = "";
118
119                     switch (message.Type)
120                     {
121                         case
ChatTwo_Protocol.MessageType.CreateUserReply:
122                             // Fire an OnCreateUserReply event.
123                             CreateUserReplyEventArgs
argsCreateUser = new CreateUserReplyEventArgs();
124                             argsCreateUser.Success =
message.Data[0] == 0x00;
125                             switch (message.Data[0])
126                             {
127                                 case 0: // Success.
128                                     break;
129                                 case 1: // Username already exist.
130                                     argsCreateUser.Message = "A
user already exist with that name.";
131                                     break;
132                             }
133                             OnCreateUserReply(argsCreateUser);
134                             break;
135                         case
ChatTwo_Protocol.MessageType.LoginReply:
136                             // Fire an OnLoginReply event.
137                             LoginReplyEventArgs argsLogin = new
LoginReplyEventArgs();
138                             argsLogin.Success = message.Data[0] ==
0x00;
139                             switch (message.Data[0])
140                             {
141                                 case 0: // Success.
142                                     int userId =
ByteHelper.ToInt32(message.Data, 1);
143                                     string username =
Encoding.Unicode.GetString(ByteHelper.SubArray(message.Data, 5));
144                                     argsLogin.Name = username;
145                                     LogIn(userId);
146                                     break;
147                                 case 1: // Wrong password.
148                                     argsLogin.Message = "Wrong
username or password.";
149                                     break;
150                                 case 2: // Already online.
151                                     argsLogin.Message = "That user
is already online.";
152                                     break;
153                             }
154                             OnLoginReply(argsLogin);
155                             break;
156                         case ChatTwo_Protocol.MessageType.Message:
157                             messageData =
ByteHelper.SubArray(args.Data, 0, 7);
158                             messageText =
Encoding.Unicode.GetString(ByteHelper.SubArray(messageBytes, 8));
```

```
159                             break;
160                         }
161                     }
162                     else
163                         throw new NotImplementedException("Could not
validate the MAC of received message.");
164                         // Need to add a simple debug message here,
but this works as a great breakpoint until then.
165                 }
166                 else
167                     throw new NotImplementedException("Could not
validate the signature of the received message. The signature was
\"0x" + args.Data[0] + "\" but only \"0x92\" is allowed.");
168                     // Need to add a simple debug message here, but
this works as a great breakpoint until then.
169             }
170
171             private static void
OnCreateUserReply(CreateUserReplyEventArgs e)
172             {
173                 EventHandler<CreateUserReplyEventArgs> handler =
CreateUserReply;
174                 if (handler != null)
175                 {
176                     handler(null, e);
177                 }
178             }
179             public static event EventHandler<CreateUserReplyEventArgs>
CreateUserReply;
180
181             private static void OnLoginReply(LoginReplyEventArgs e)
182             {
183                 EventHandler<LoginReplyEventArgs> handler =
LoginReply;
184                 if (handler != null)
185                 {
186                     handler(null, e);
187                 }
188             }
189             public static event EventHandler<LoginReplyEventArgs>
LoginReply;
190
191             public static void
MessageToServer(ChatTwo_Protocol.MessageType type, byte[] data = null,
string text = null)
192             {
193                 Message message = new Message();
194                 message.From = _user.ID;
195                 message.To = ChatTwo_Protocol.ServerReserrvedUserID;
196                 message.Type = type;
197                 if (data != null && data.Length != 0)
198                     message.Data = data;
199                 if (!String.IsNullOrEmpty(text))
200                     message.Text = text;
201                 message.Ip = _serverAddress;
202                 MessageTransmissionHandler(message);
203             }
204
```

```
205         public static void MessageToUser(int to,
ChatTwo_Protocol.MessageType type, byte[] data = null, string text =
null)
206         {
207             Message message = new Message();
208             message.From = _user.ID;
209             message.To = to;
210             message.Type = type;
211             if (data != null && data.Length != 0)
212                 message.Data = data;
213             if (!String.IsNullOrEmpty(text))
214                 message.Text = text;
215             if (_contacts.Any(x => x.ID == to))
216                 message.Ip = _contacts.Find(x => x.ID ==
to).Socket;
217             MessageTransmissionHandler(message);
218         }
219
220         public static void MessageTransmissionHandler(Message
message)
221         {
222             byte[] messageBytes =
ChatTwo_Protocol.MessageTransmissionHandler(message);
223
224             string sharedSecret;
225             if (message.Type ==
ChatTwo_Protocol.MessageType.CreateUser)
226             {
227                 sharedSecret =
ChatTwo_Protocol.DefaultSharedSecret;
228             }
229             else if (message.Type ==
ChatTwo_Protocol.MessageType.Login)
230             {
231                 ServerSharedSecret =
ByteHelper.GetHashString(messageBytes);
232                 sharedSecret = ServerSharedSecret;
233             }
234             else if (message.To ==
ChatTwo_Protocol.ServerReserrvedUserID)
235             {
236                 sharedSecret = ServerSharedSecret;
237             }
238             else
239             {
240                 int userId = message.To;
241                 sharedSecret = _contacts.Find(x => x.ID ==
userId).Secret;
242             }
243
244             messageBytes =
ChatTwo_Protocol.AddSignatureAndMac(messageBytes, sharedSecret);
245
246             // Fire an OnMessageTransmission event.
247             PacketTransmissionEventArgs args = new
PacketTransmissionEventArgs();
248             args.Destination = message.Ip;
249             args.PacketContent = messageBytes;
```

```
250             OnMessageTransmission(args);
251         }
252
253         private static void
OnMessageTransmission(PacketTransmissionEventArgs e)
254         {
255             EventHandler<PacketTransmissionEventArgs> handler =
MessageTransmission;
256             if (handler != null)
257             {
258                 handler(null, e);
259             }
260         }
261         public static event
EventHandler<PacketTransmissionEventArgs> MessageTransmission;
262     }
263
264     public class CreateUserReplyEventArgs : EventArgs
265     {
266         public bool Success { get; set; }
267         public string Message { get; set; }
268     }
269
270     public class LoginReplyEventArgs : EventArgs
271     {
272         public bool Success { get; set; }
273         public string Name { get; set; }
274         public string Message { get; set; }
275     }
276 }
```

Shared Code

ByteHelper.cs

```
001 using System;
002 using System.Collections.Generic;
003 using System.Linq;
004 using System.Text;
005 using System.Security.Cryptography;
006
007 namespace ChatTwo_Server
008 {
009     class ByteHelper
010     {
011         /// <summary>
012         /// Converts a byte array to a Hexadecimal string.
013         /// </summary>
014         /// <param name="singleByte">Byte to be converted.</param>
015         static public string ToHex(byte singleByte) // Based on
http://stackoverflow.com/a/10048895
016         {
017             char[] hex = new char[2];
018
019             byte b;
020
021             b = ((byte)(singleByte >> 4));
```

```
022             hex[0] = (char)(b > 9 ? b - 10 + 'A' : b + '0');
023
024             b = ((byte)(singleByte & 0x0F));
025             hex[1] = (char)(b > 9 ? b - 10 + 'A' : b + '0');
026
027             return new string(hex);
028         }
029         /// <summary>
030         /// Converts a byte array to a Hexadecimal string.
031         /// </summary>
032         /// <param name="bytes">Bytes to be converted.</param>
033         static public string ToHex(byte[] bytes)
034         {
035             List<string> hexs = new List<string>();
036             foreach (byte singleByte in bytes)
037                 hexs.Add(ToHex(singleByte));
038             return string.Join("-", hexs.ToArray());
039         }
040
041         /// <summary>
042         /// Converts four bytes from a byte array to a int32.
043         /// </summary>
044         /// <param name="bytes">Bytes to be converted.</param>
045         /// <param name="startIndex">Index of the starting
byte.</param>
046         static public int ToInt32(byte[] bytes, int startIndex)
047         {
048             if (!BitConverter.IsLittleEndian)
049             {
050                 bytes = ByteHelper.SubArray(bytes, startIndex, 4);
051                 Array.Reverse(bytes);
052                 startIndex = 0;
053             }
054             return BitConverter.ToInt32(bytes, startIndex);
055         }
056
057         /// <summary>
058         /// Converts four bytes from a byte array to a float.
059         /// </summary>
060         /// <param name="bytes">Bytes to be converted.</param>
061         /// <param name="startIndex">Index of the starting
byte.</param>
062         static public float ToFloat(byte[] bytes, int startIndex)
063         {
064             byte[] subBytes = ByteHelper.SubArray(bytes,
startIndex, 4);
065             if (BitConverter.IsLittleEndian)
066                 Array.Reverse(subBytes);
067             return BitConverter.ToSingle(subBytes, 0);
068         }
069
070         /// <summary>
071         /// Converts a byte array to a string, using
BigEndianUnicode.
072         /// </summary>
073         /// <param name="bytes">Bytes to be converted.</param>
074         /// <param name="startIndex">Index of the starting
byte.</param>
```

```
075         /// <param name="length">Number of bytes to
convert.</param>
076         static public string ToBigEndianUnicodeString(byte[]
bytes, int startIndex, int length)
077         {
078             byte[] subBytes = ByteHelper.SubArray(bytes,
startIndex, length);
079             //subBytes = Helper.ConcatinateArray(new byte[] {
0xFE, 0xFF }, subBytes);
080             //if (BitConverter.IsLittleEndian)
081             //    Array.Reverse(subBytes);
082             string text =
Encoding.BigEndianUnicode.GetString(subBytes); // UTF-16 BigEndian to
string.
083             return text;
084         }
085
086         /// <summary>
087         /// Returns the Hexavigesimal letters only ID.
088         /// </summary>
089         /// <param name="numberID">Int32 version of the
ID.</param>
090         static public string ToID(int numberID)
091         { // http://en.wikipedia.org/wiki/Hexavigesimal
092             numberID = Math.Abs(numberID);
093             String converted = "";
094             // Repeatedly divide the number by 26 and convert the
095             // remainder into the appropriate letter.
096             while (numberID > 0)
097             {
098                 int remainder = (numberID) % 26;
099                 converted = converted + (char)(remainder + 'A');
100                 numberID = (numberID - remainder) / 26;
101             }
102
103             return converted;
104         }
105
106         public const int HashByteLength = 20;
107         /// <summary>
108         /// Returns the SHA1 hash of a byte array in a
Base64String.
109         /// </summary>
110         /// <param name="numberID">Byte array to be
hashed.</param>
111         static public string GetHashString(byte[] bytes)
112         {
113             string hash =
Convert.ToBase64String(GetHashBytes(bytes));
114             return hash;
115         }
116
117         /// <summary>
118         /// Returns the SHA1 hash of a byte array in a byte array.
119         /// </summary>
120         /// <param name="numberID">Byte array to be
hashed.</param>
121         static public byte[] GetHashBytes(byte[] bytes)
```

```
122         {
123             byte[] hash;
124             using (SHA1CryptoServiceProvider sha1 = new
SHA1CryptoServiceProvider())
125             {
126                 hash = sha1.ComputeHash(bytes);
127             }
128             return hash;
129         }
130 
131         /// <summary>
132         /// Returns selected part of a byte array.
133         /// </summary>
134         /// <param name="bytes">Full byte array.</param>
135         /// <param name="startIndex">Index of the starting
byte.</param>
136         static public byte[] SubArray(byte[] bytes, int
startIndex)
137         {
138             if (startIndex == 0)
139                 return bytes;
140             return SubArray(bytes, startIndex, bytes.Length -
startIndex);
141         }
142         /// <summary>
143         /// Returns selected part of a byte array.
144         /// </summary>
145         /// <param name="bytes">Full byte array.</param>
146         /// <param name="startIndex">Index of the starting
byte.</param>
147         /// <param name="length">Number of bytes to
return.</param>
148         static public byte[] SubArray(byte[] bytes, int
startIndex, int length)
149         {
150             byte[] rv = new byte[length];
151             System.Buffer.BlockCopy(bytes, startIndex, rv, 0,
length);
152             return rv;
153             //return new List<byte>(bytes).GetRange(startIndex,
length).ToArray(); // Another ways of doing it.
154         }
155 
156         /// <summary>
157         /// Combine multiple arrays into one.
158         /// One after the other.
159         /// </summary>
160         static public byte[] ConcatinateArray(byte[] array1,
byte[] array2)
161         {
162             byte[] rv = new byte[array1.Length + array2.Length];
163             System.Buffer.BlockCopy(array1, 0, rv, 0,
array1.Length);
164             System.Buffer.BlockCopy(array2, 0, rv, array1.Length,
array2.Length);
165             return rv;
166         }
167         /// <summary>
```

```
168         /// Combine multiple arrays into one.
169         /// One after the other.
170         /// </summary>
171         static public byte[] ConcatinateArray(byte[] array1,
byte[] array2, byte[] array3)
172         {
173             byte[] rv = new byte[array1.Length + array2.Length +
array3.Length];
174             System.Buffer.BlockCopy(array1, 0, rv, 0,
array1.Length);
175             System.Buffer.BlockCopy(array2, 0, rv, array1.Length,
array2.Length);
176             System.Buffer.BlockCopy(array3, 0, rv, array1.Length +
array2.Length, array3.Length);
177             return rv;
178         }
179
180         /// <summary>
181         /// Returns true if flag is in bitCode.
182         /// ((bitCode AND flag) == flag)
183         /// </summary>
184         /// <param name="bitCode">Byte or int32 used for bit
flags.</param>
185         /// <param name="flag">Flag to check for.</param>
186         static public bool FlagCheck(int bitCode, int flag)
187         {
188             return ((bitCode & flag) == flag);
189         }
190
191         /// <summary>
192         /// Returns a string that is void of HTML tags.
193         /// Attempts to add newlines where needed.
194         /// </summary>
195         /// <param name="input">HTML string.</param>
196         static public string CleanText(string input) // Removes
the html code tags.
197         {
198             if (input.Contains("<") && input.Contains(">"))
199             {
200                 int tagStart, tagEnd;
201                 string processed = "";
202                 while (input.Contains("<") && input.Contains(">"))
203                 {
204                     tagStart = input.IndexOf('<');
205                     tagEnd = input.IndexOf('>') - tagStart;
206                     processed += input.Remove(tagStart);
207                     string tag = input.Substring(tagStart + 1,
tagEnd - 1);
208                     input = input.Substring(tagStart + tagEnd +
1);
209                     switch (tag.Split(new char[] { ' ' },
StringSplitOptions.RemoveEmptyEntries)[0])
210                     {
211                         case "br":
212                         case "div":
213                         case "/div":
214                         case "/td":
215                         case "/tr":
```

```
216                             processed += Environment.NewLine;
217                             break;
218                         //case "b":
219                         //case "/b":
220                         //case "td":
221                         //case "tr":
222                         //    break;
223                     }
224                 }
225                 processed = processed.Replace(Environment.NewLine
+ Environment.NewLine + Environment.NewLine, Environment.NewLine); //
Remove triple NewLines.
226                 processed = processed.Replace(" ", " "); //
Replace " " with a normal " ".
227                 return processed.Trim(); // Trim for good measure
before reuturning the text.
228             }
229             else
230                 return input.Trim();
231         }
232     }
233 }
```

UserObj.cs

```
01 using System;
02 using System.Collections.Generic;
03 using System.Linq;
04 using System.Text;
05 using System.Net;
06 using System.Globalization;
07
08 namespace ChatTwo_Server
09 {
10
11     public class UserObj
12     {
13         public int ID { set; get; }
14         public string Name { set; get; }
15         public bool Online { set; get; }
16         public IPEndPoint Socket { set; get; }
17         public DateTime LastOnline { set; get; }
18         public DateTime Registered { set; get; }
19         public string Secret { set; get; }
20
21         public UserObj()
22         {
23         }
24
25         public void StringSocket(string socket)
26         {
27             if (String.IsNullOrEmpty(socket))
28                 Socket = null;
29             else
30                 Socket = CreateIPEndPoint(socket);
31         }
32
33         public override string ToString()
```

```
34          {
35              return "user[" + ID + "] Name: " + Name +
Environment.NewLine +
36                     "user[" + ID + "] Online: " + Online +
Environment.NewLine +
37                     "user[" + ID + "] Socket: " + Socket +
Environment.NewLine +
38                     "user[" + ID + "] LastOnline: " +
LastOnline.ToString("yyyy-MM-dd HH:mm:ss") + Environment.NewLine +
39                     "user[" + ID + "] Registered: " +
Registered.ToString("yyyy-MM-dd HH:mm:ss");
40          }
41
42          // Handles IPv4 and IPv6 notation.
43          // http://stackoverflow.com/questions/2727609/best-way-to-
create-ipendpoint-from-string
44          private IPEndPoint CreateIPEndPoint(string endPoint)
45          {
46              string[] ep = endPoint.Split(':');
47              if (ep.Length < 2) throw new FormatException("Invalid
endpoint format");
48              IPAddress ip;
49              if (ep.Length > 2)
50              {
51                  if (!IPAddress.TryParse(string.Join(":", ep, 0,
ep.Length - 1), out ip))
52                  {
53                      throw new FormatException("Invalid ip-adress");
54                  }
55              }
56              else
57              {
58                  if (!IPAddress.TryParse(ep[0], out ip))
59                  {
60                      throw new FormatException("Invalid ip-adress");
61                  }
62              }
63              int port;
64              if (!int.TryParse(ep[ep.Length - 1], NumberStyles.None,
NumberFormatInfo.CurrentInfo, out port))
65              {
66                  throw new FormatException("Invalid port");
67              }
68              return new IPEndPoint(ip, port);
69          }
70      }
71  }
```

ChatTwo_Protocol.cs

```
001 using System;
002 using System.Collections.Generic;
003 using System.Linq;
004 using System.Text;
005 using System.Net;
006 using System.Net.Sockets;
007
008 namespace ChatTwo_Server
```

```
009 {
010     public static class ChatTwo_Protocol
011     {
012         const byte _version = 0x00;
013 
014         public const int SignatureByteLength = 2;
015 
016         public const int ServerReserrvedUserID = 0;
017 
018         public const string DefaultSharedSecret =
"5ny1mzFo4S6nh7hDcqsHVg+DBNU=";
019 
020         public enum MessageType
021         {
022             CreateUser, // When a new user is joining the server,,
creating a username and password.
023             CreateUserReply, // Reply of success or failure of
user creation.
024             Login, // Login attempt.
025             LoginReply, // Login attempt response.
026             Status, // Tell server your online status and IP
address. A form of keepalive.
027             ContactRequest, // A request to make someone your
contact.
028             ContactRevoke, // Remove someone from your contacts.
029             ContactStatus, // Tell client the online status and IP
address of a contact.
030             Message, // Message to another user.
031             RelayMessage, // Request for the server to relay a
message to another user. Used if peer-to-peer fail?
032             Logout // Terminate all communication.
033         }
034 
035         public static bool ValidateMac(byte[] bytes, string
sharedSecret)
036         {
037             string mac = Convert.ToBase64String(bytes, 2,
ByteHelper.HashByteLength);
038 #if DEBUG
039             string test = CreateMac(ByteHelper.SubArray(bytes,
SignatureByteLength + ByteHelper.HashByteLength), sharedSecret);
040 #endif
041             bool macValid = CreateMac(ByteHelper.SubArray(bytes,
SignatureByteLength + ByteHelper.HashByteLength), sharedSecret) ==
mac;
042             return macValid;
043         }
044 
045         public static byte[] AddSignatureAndMac(byte[] bytes,
string sharedSecret)
046         {
047             TimeSpan sinceMidnight = DateTime.Now -
DateTime.Today;
048             int timez = (int)sinceMidnight.TotalMilliseconds;
049             bytes =
ByteHelper.ConcatinateArray(BitConverter.GetBytes(timez), bytes); //
Add a milisecond timestamp to the meassage.
050 
```

```
051             byte[] macBytes =
Convert.FromBase64String(CreateMac(bytes, sharedSecret));
052
053             byte[] singatureBytes = new byte[] { 0x92, _version };
// Signature byte and version byte.
054
055             bytes = ByteHelper.ConcatinateArray(singatureBytes,
macBytes, bytes);
056             return bytes;
057         }
058
059         public static byte[] RemoveSignatureAndMac(byte[] bytes)
060         {
061             bytes = ByteHelper.SubArray(bytes, SignatureByteLength
+ ByteHelper.HashByteLength); // Remove the signature, the version
number and the MAC.
062             return bytes;
063         }
064
065         private static string CreateMac(byte[] messageBytes,
string sharedSecret)
066         {
067             return
ByteHelper.GetHashString(ByteHelper.ConcatinateArray(ByteHelper.GetHas
hBytes(messageBytes), Convert.FromBase64String(sharedSecret)));
068         }
069
070         public static Message
MessageReceivedHandler(PacketReceivedEventArgs args)
071         {
072             args.Data =
ChatTwo_Protocol.RemoveSignatureAndMac(args.Data);
073
074             Message messageObj = new Message();
075             messageObj.Ip = args.Sender;
076             int milliseconds = ByteHelper.ToInt32(args.Data, 0);
077             messageObj.Timez = String.Format("{0}:{1}:{2}",
(milliseconds / (60 * 60 * 1000)) % 24, (milliseconds / (60 * 1000)) %
60, (milliseconds / (1000)) % 60);
078             messageObj.From = ByteHelper.ToInt32(args.Data, 4);
079             messageObj.Type = (MessageType)args.Data[8];
080             messageObj.Data = ByteHelper.SubArray(args.Data, 9);
081
082             return messageObj;
083         }
084
085         public static byte[] MessageTransmissionHandler(Message
message)
086         {
087             byte[] textBytes = new byte[0];
088             if (!String.IsNullOrEmpty(message.Text))
089                 textBytes =
Encoding.Unicode.GetBytes(message.Text);
090
091             byte[] dataBytes = new byte[0];
092             if (message.Data != null)
093                 dataBytes = message.Data;
094
```

```
095             byte[] messageBytes =
ByteHelper.ConcatinateArray(BitConverter.GetBytes(message.From), new
byte[] { (byte)message.Type });
096             messageBytes =
ByteHelper.ConcatinateArray(messageBytes, dataBytes, textBytes);
097
098             return messageBytes;
099         }
100
101         private static void
OnMessageTransmission(PacketTransmissionEventArgs e)
102         {
103             EventHandler<PacketTransmissionEventArgs> handler =
MessageTransmission;
104             if (handler != null)
105             {
106                 handler(null, e);
107             }
108         }
109         public static event
EventHandler<PacketTransmissionEventArgs> MessageTransmission;
110     }
111
112     public class Message
113     {
114         public int To { get; set; }
115         public int From { get; set; }
116         public IPEndPoint Ip { get; set; }
117         public ChatTwo_Protocol.MessageType Type { get; set; }
118         public string Timez { get; set; }
119         public byte[] Data { get; set; }
120         public string Text { get; set; }
121     }
122 }
```

IpCommunication.cs

```
001 using System;
002 using System.Collections.Generic;
003 using System.Linq;
004 using System.Text;
005 using System.Net;
006 using System.Net.Sockets;
007 using System.Threading;
008
009 namespace ChatTwo_Server
010 {
011     class IpCommunication
012     {
013         protected bool _online;
014         public bool Active
015         {
016             get { return _online; }
017         }
018
019         protected virtual void
OnPacketReceived(PacketReceivedEventArgs e)
020         {
```

```
021             EventHandler<PacketReceivedEventArgs> handler =
PacketReceived;
022             if (handler != null)
023             {
024                 handler(this, e);
025             }
026         }
027
028         public event EventHandler<PacketReceivedEventArgs>
PacketReceived;
029     }
030
031     class UdpCommunication : IpCommunication
032     {
033         protected Thread _threadPacketListener;
034         protected Thread _threadPacketSending;
035
036         protected UdpClient _client;
037         public UdpClient Client
038         {
039             get { return _client; }
040             set { _client = value; }
041         }
042
043         public int Port
044         {
045             get { return
((IPEndPoint)_client.Client.LocalEndPoint).Port; }
046         }
047
048         protected List<ControlledPacket>
_messageSendingControlList = new List<ControlledPacket>();
049         protected List<string> _messageReceivingControlList = new
List<string>();
050
051         /// <summary>
052         /// Creates a temp UdpClient and sends an
EtherConnectionTest packet to the target address.
053         /// </summary>
054         /// <param name="address">Target address for the
EtherConnectionTest packet.</param>
055         public static bool TestPortforward(IPEndPoint address)
056         {
057             //// Check if the port number is in use.
058             //bool isInUse =
System.Net.NetworkInformation.IPGlobalProperties.GetIPGlobalProperties
().GetActiveUdpListeners().Any(p => p.Port == port);
059
060             // Rather than just checking if the portnumber is in
use, which only causes "new UdpClient(port)" to fail, I want to test
if the server can hit it self by pinging the external IP address.
061             try
062             {
063                 using (UdpClient tempClient = new UdpClient(0))
064                 {
065                     #region tempClient.Client.IOControl // Windows
UDP Bugfix
```

```
066                     // This is a fix to make the UdpClient ignore
some weird behavior from Windows.
067                     // Read more here:
http://stackoverflow.com/a/7478498
068                     const uint IOC_IN = 0x80000000;
069                     const uint IOC_VENDOR = 0x18000000;
070                     uint SIO_UDP_CONNRESET = IOC_IN | IOC_VENDOR |
12;
071
tempClient.Client.IOControl((int)SIO_UDP_CONNRESET, new byte[] {
Convert.ToByte(false) }, null);
072                     #endregion
073                     byte[] messageBytes = new byte[] { 0xEC };
074                     tempClient.Send(messageBytes,
messageBytes.Length, address); // Send the message.
075                 }
076             }
077             catch (SocketException ex)
078             {
079                 System.Diagnostics.Debug.WriteLine("### An error
happened when trying to send out an EtherConnectionTest packet:"); //
Called it "EtherConnection" because 0xEC was a nice hex value.
080                 System.Diagnostics.Debug.WriteLine("### " +
ex.Message);
081                 System.Diagnostics.Debug.WriteLine("### " +
ex.ToString());
082                 return false;
083             }
084             return true;
085         }
086
087         protected void OnEtherConnectionReply(EventArgs e)
088         {
089             EventHandler<EventArgs> handler =
EtherConnectionReply;
090             if (handler != null)
091             {
092                 handler(this, e);
093             }
094         }
095
096         public event EventHandler<EventArgs> EtherConnectionReply;
097
098         /// <summary>
099         /// Starts the UdpClient and the threaded methods.
100         /// </summary>
101         /// <param name="serverPort">Port number the UdpClient
should use. 0 (zero) will let the OS choose a random port.</param>
102         public bool Start(int serverPort)
103         {
104             try
105             {
106                 _client = new UdpClient(serverPort);
107                 _client.Client.ReceiveTimeout = 1000; // This
causes the _client.Receive(ref remoteSender) methtod to actually
timeout, else it would simply freeze the _threadPacketListener thread.
108                 #region _client.Client.IOControl // Windows UDP
Bugfix
```

```
109                 // This is a fix to make the UdpClient ignore some
weird behavior from Windows.
110                 // Read more here:
http://stackoverflow.com/a/7478498
111                 const uint IOC_IN = 0x80000000;
112                 const uint IOC_VENDOR = 0x18000000;
113                 uint SIO_UDP_CONNRESET = IOC_IN | IOC_VENDOR | 12;
114                 _client.Client.IOControl((int)SIO_UDP_CONNRESET,
new byte[] { Convert.ToByte(false) }, null);
115                 #endregion
116             }
117             catch (SocketException ex)
118             {
119                 System.Diagnostics.Debug.WriteLine("### Starting
the UdpClient on port \"" + serverPort + "\" failed:");
120                 System.Diagnostics.Debug.WriteLine("### " +
ex.Message);
121                 System.Diagnostics.Debug.WriteLine("### " +
ex.ToString());
122                 return false;
123             }
124             _online = true;
125             _threadPacketListener = new Thread(new
ThreadStart(ReceivePacket));
126             _threadPacketListener.Name = "Packet Listening Thread
(ReceivePacket method)";
127             _threadPacketListener.Start();
128             _threadPacketSending = new Thread(new
ThreadStart(PacketTransmissionControl));
129             _threadPacketSending.Name = "Packet Sending Thread
(PacketTransmissionControl method)";
130             _threadPacketSending.Start();
131             return true;
132         }
133
134         /// <summary>
135         /// Stops all threaded methods and stops the UdpClient.
Use this before closing the application!
136         /// </summary>
137         public void Stop()
138         {
139             if (_online)
140             {
141                 _online = false;
142                 //_threadPacketListener.Abort(); // This caused
some problems.
143                 _threadPacketListener.Join(); // Wait for
_threadPacketListener's next "am I online?" check.
144                 //_threadPacketSending.Abort(); // This caused
some problems.
145                 _threadPacketSending.Join(); // Wait for
_threadPacketSending's next "am I online?" check.
146                 _client.Close();
147             }
148         }
149
150         /// <summary>
151         /// Create an ACK packet from a base64 hash string.
```

```
152         /// </summary>
153         /// <param name="hash">Base64 hash string to be
used.</param>
154         protected byte[] CreateAck(string hash)
155         {
156             byte[] ackTag = new byte[] { 0xCE }; // 0xCE = 206
157             byte[] ackBytes = ByteHelper.ConcatinateArray(ackTag,
Convert.FromBase64String(hash), ackTag);
158             return ackBytes;
159         }
160 
161         /// <summary>
162         /// Convert an ACK packet to a base64 hash string.
163         /// </summary>
164         /// <param name="packetBytes">ACK packet's byte
content.</param>
165         protected string OpenAck(byte[] packetBytes)
166         {
167             string ackHash = Convert.ToBase64String(packetBytes,
1, ByteHelper.HashByteLength);
168             return ackHash;
169         }
170 
171         /// <summary>
172         /// This is a threaded method that keeps looping while
_online is true.
173         /// It will receive UDP messages on the UdpClient's port
number and forward them to the OnPacketReceived event.
174         /// </summary>
175         public void ReceivePacket() // Threaded looping method.
176         {
177             while (_online)
178             {
179                 try
180                 {
181                     IPEndPoint remoteSender = new
IPEndPoint(IPAddress.Any, 0);
182                     byte[] receivedBytes = _client.Receive(ref
remoteSender);
183                     if (receivedBytes != null &&
receivedBytes.Length != 0)
184                     {
185                         if (receivedBytes.Length ==
ByteHelper.HashByteLength + 2 && receivedBytes[0] == 0xCE &&
receivedBytes[receivedBytes.Length - 1] == 0xCE)
186                         {
187                             if (_messageSendingControlList.Count
!= 0)
188                             {
189                                 // The received message is a ACK
message.
190                                 string hash =
OpenAck(receivedBytes);
191                                 ControlledPacket packet =
_messageSendingControlList.Find(x => x.Hash == hash);
192                                 if (packet != null)
193 
_messageSendingControlList.Remove(packet);
```

```
194                             }
195                         }
196                         else if (receivedBytes.Length == 1 &&
receivedBytes[0] == 0xEC)
197                         {
198                             // Fire an OnEtherConnectionReply
event.
199                             OnEtherConnectionReply(null);
200                         }
201                         else
202                         {
203                             // Send back an ACK packet.
204                             string hash =
ByteHelper.GetHashString(receivedBytes);
205                             byte[] ackBytes = CreateAck(hash);
206                             _client.Send(ackBytes,
ackBytes.Length, remoteSender);
207
208                             // Check if the message is a
duplicate.
209                             if
(!_messageReceivingControlList.Any(x => x == hash))
210                             {
211                                 // Add the message's hash to a
list so we don't react on the same message twice.
212
_messageReceivingControlList.Add(hash);
213                                 if
(_messageReceivingControlList.Count > 5) // Only keep the latest 5
messages.
214
_messageReceivingControlList.RemoveAt(0);
215
216                                 // Fire an OnPacketReceived event.
217                                 PacketReceivedEventArgs args = new
PacketReceivedEventArgs();
218                                 args.Sender = remoteSender;
219                                 args.Data = receivedBytes;
220                                 OnPacketReceived(args);
221                             }
222                         }
223                     }
224                 }
225                 catch (SocketException ex)
226                 {
227                     if (ex.SocketErrorCode !=
SocketError.TimedOut)
228                     {
229                         System.Diagnostics.Debug.WriteLine("### "
+ _threadPacketListener.Name + " has crashed:");
230                         System.Diagnostics.Debug.WriteLine("### "
+ ex.Message);
231                         System.Diagnostics.Debug.WriteLine("### "
+ ex.ToString());
232                         break;
233                     }
234                     else
235                         continue;
```

```
236                 }
237             }
238         }
239 
240         /// <summary>
241         /// Send a packet to a target IP address.
242         /// </summary>
243         /// <param name="sender">Default object parameter for
event receiving methods. Unused here.</param>
244         /// <param name="args">PacketTransmissionEventArgs object
containing the byte array to be send and the destination IP
address.</param>
245         public void SendPacket(object sender,
PacketTransmissionEventArgs args)
246         {
247             ControlledPacket ctrlPacket = new ControlledPacket();
248             ctrlPacket.Recipient = args.Destination;
249             ctrlPacket.Data = args.PacketContent;
250 
251             _messageSendingControlList.Add(ctrlPacket);
252         }
253 
254         /// <summary>
255         /// This is a threaded method that keeps looping while
 online is true.
256         /// It will check the _packetSendingControlList list and
try to send all packets on the list 5 times per second.
257         /// </summary>
258         protected void PacketTransmissionControl() // Threaded
looping method.
259         {
260             try
261             {
262                 while (_online)
263                 {
264                     CheckPacketControlList();
265                     Thread.Sleep(200);
266                 }
267             }
268             catch (Exception ex)
269             {
270                 System.Diagnostics.Debug.WriteLine("### " +
_threadPacketSending.Name + " has crashed:");
271                 System.Diagnostics.Debug.WriteLine("### " +
ex.Message);
272                 System.Diagnostics.Debug.WriteLine("### " +
ex.ToString());
273             }
274         }
275 
276         protected void CheckPacketControlList()
277         {
278             if (_messageSendingControlList.Count != 0)
279             {
280                 List<ControlledPacket> temp =
_messageSendingControlList.FindAll(x => (x.LastTry == null ||
(DateTime.Now - x.LastTry).TotalMilliseconds > 400) && x.Attempts <
5);
```

```
281                 foreach (ControlledPacket ctrlPacket in temp)
282                 {
283                     if (SendControlledPacket(ctrlPacket))
284                     {
285                         ctrlPacket.LastTry = DateTime.Now;
286                         ctrlPacket.Attempts++;
287 #if !DEBUG
288                         if (ctrlPacket.Attempts == 5)
289 
_messageSendingControlList.Remove(ctrlPacket);
290 #endif
291                     }
292                 }
293             }
294         }
295 
296         /// <summary>
297         /// Attempt to send the controlled packet. Return true if
successful.
298         /// </summary>
299         /// <param name="ctrlPacket">Packet to be sent.</param>
300         protected bool SendControlledPacket(ControlledPacket
ctrlPacket)
301         {
302             try
303             {
304                 _client.Send(ctrlPacket.Data,
ctrlPacket.Data.Length, ctrlPacket.Recipient); // Send the packet.
305             }
306             catch (SocketException ex)
307             {
308 #if DEBUG
309                 throw;
310 #else
311                 System.Diagnostics.Debug.WriteLine("### An error
happened when trying to send ControlledPacket:");
312                 System.Diagnostics.Debug.WriteLine("### " +
ex.Message);
313                 System.Diagnostics.Debug.WriteLine("### " +
ex.ToString());
314                 return false;
315 #endif
316             }
317             return true;
318         }
319     }
320 
321     public class PacketReceivedEventArgs : EventArgs
322     {
323         public IPEndPoint Sender { get; set; }
324         public byte[] Data { get; set; }
325     }
326 
327     public class PacketTransmissionEventArgs : EventArgs
328     {
329         public IPEndPoint Destination { get; set; }
330         public byte[] PacketContent { get; set; }
331     }
```

```
332
333     internal class ControlledPacket
334     {
335         public IPEndPoint Recipient { get; set; }
336         public byte[] Data { get; set; }
337         public string Hash { get { return
ByteHelper.GetHashString(Data); } }
338         public DateTime LastTry { get; set; }
339         public int Attempts { get; set; }
340     }
341 }
```