School IT Classroom Management Software

SOFT 050 Computing Software Project Report

Dan Scott – Plymouth University Undergraduate

2015

Table of Contents

[Introduction 1](#_Toc417651996)

[Implementation 1](#_Toc417651997)

[Design 5](#_Toc417651998)

[Review 6](#_Toc417651999)

[What have I learnt from this project? 6](#_Toc417652000)

[What I did well 6](#_Toc417652001)

[What could I have added if I had more time 6](#_Toc417652002)

[Project management 6](#_Toc417652003)

[References 6](#_Toc417652004)

# Introduction

For my Software Project I decided to develop some school IT Classroom management software. The project utilises network programming using the .NET framework, most notably the Socket class of the framework. It’s designed to run on windows based operating systems as the .NET framework as of this time of writing the report is not cross platform, however it has recently gone open source so here’s to hoping for the future! The functionality that I hoped to achieve for this project was to be able to see the screens of every computer connected to the server updated in real time as to what’s on the computer screen, be able to lock a selected computer or every computer, be able to send messages to and from the teacher to student computer, launch applications that are installed on the client machine from the teacher machine.

I chose to do this project as I feel like I am quite proficient with the C# language as it is however the .NET framework is extensive enough to allow me to learn something new every day. As I am developing with the .NET framework, naturally I am going to be using Microsoft Visual Studio to develop my application. As I am going to be making a server and client I am hoping to make two different applications being a client and a server both using a common Dynamic Link Library that has generic code that both applications use to cut down on code repetition.

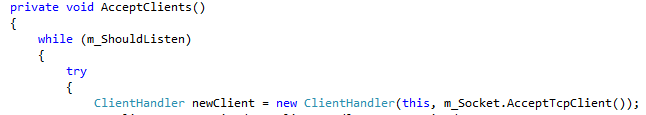
To ensure that my project didn’t get lost or broken beyond repair, I used GitHub as my version control system. This allowed me to also be able to document my development progress as well as being able to keep my source code safe.

# Implementation

My initial architecture of my application was done exactly as I had originally planned which was a client application, server application and a common DLL. However I started designing this with console applications so I didn’t have to worry about the user interface. I did this so I could just focus on getting the core logic of my application down before delving further into the user experience of it.

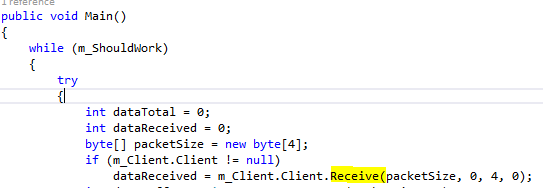
I had no prior experience to network programming so I used this project as an excuse to learn the concepts behind network programming. I had no idea of how to go about the project so before going ahead with the programming I did quite a lot of research on the MSDN archives around network programming. I looked at Asynchronous and Synchronous Socket methods separately and played around with console applications to accept connections and send strings over them. In the end I decided to go with the Synchronous method so that once a socket connection was accepted there was no need to re-instantiate the class or socket connection to send something else. This method was also seemingly more reliable at keeping the connections alive and receiving information in real time.

Once I decided that Synchronous was the method I was going to use, I started to look at various code examples of Synchronous socket communication for sending text to and from the Client and Server which is also when I had my first look at Threading in the .NET framework. I had to use Threading for this as I had a listener class that bound itself to a port and waited for a socket to connect to it, which is what is called a blocking function that stops the execution of the program until a socket has been accepted.

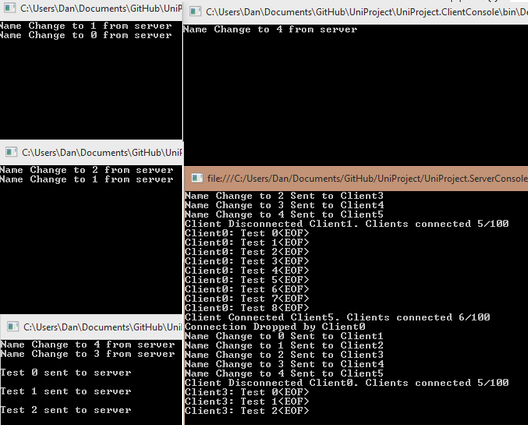


Blocking Function

Once the socket connection was accepted, the program then instantiates a new instance of the ClientHandler class which gets passed the socket that has just been accepted. The ClientHandler class then creates another thread to handle the data being transmitted received over the socket as it needs to be able to wait for data blocking the execution of the thread, whereas the Send function does not need to be blocking.

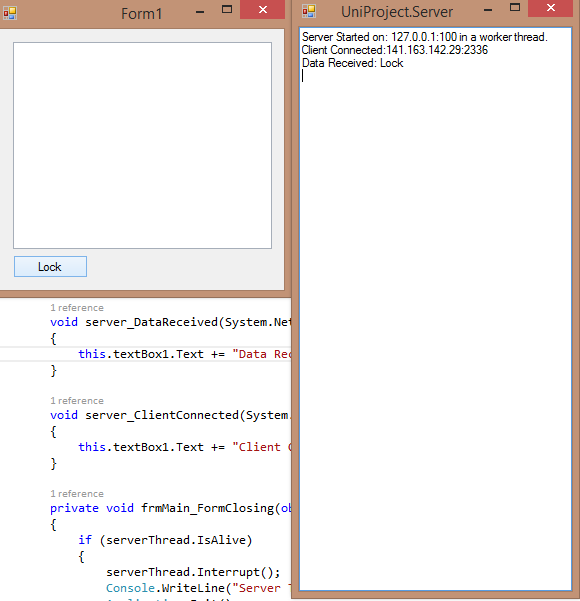


Blocking Function

In the initial pass of my server architecture, the receive function waited for an “<EOF>” statement to stop sending the data. As you can see from the screenshot below

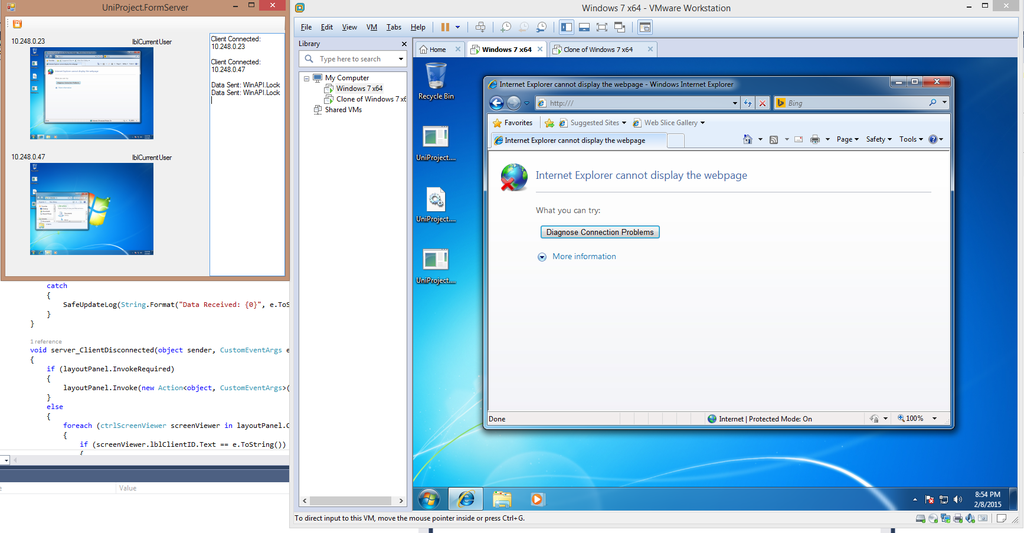
The way in which I tell the server class that data has been received is via event handlers that I defined myself along with the delegate methods to store the information that is being sent to and from the server. This data is held within the EventArgs that again I defined myself and was then able to display on the Server Window within the Event Handler method. I feel like there may be a better way in which to do this but I found that this implementation was easy to achieve and to allow other classes to subscribe to the event.

Later on in development I found that if I stored the total amount of bytes to send, and then told the send function to stop sending packets once the total about of bytes has been sent then I could send any file format, not just text. Which is when my project really moved away from console programs and then moved on to the WinForms design process. Essentially replicating my Console Application in WinForms was a good starting point as you can see here:



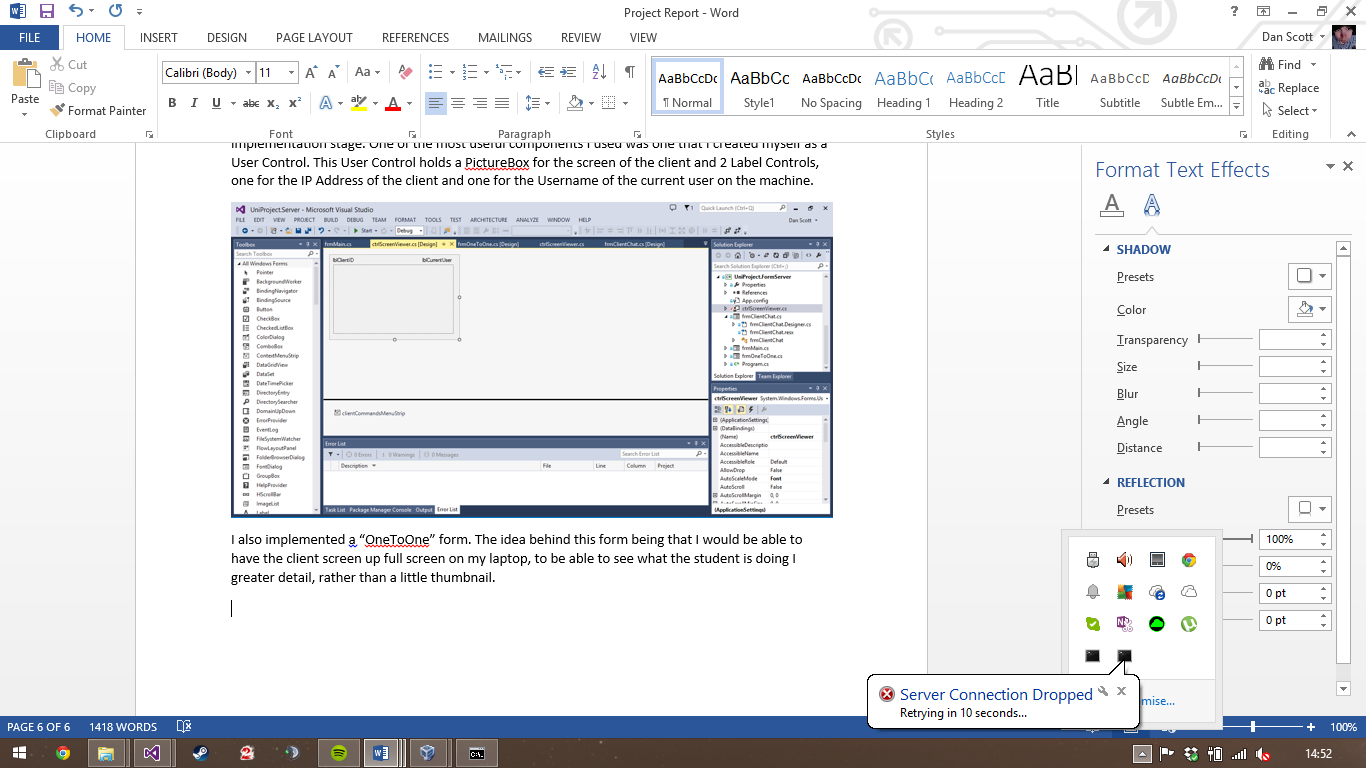
But this is also where I found that my initial idea of the server being on the student computer maybe wasn’t such a good idea as I would previously have to know the IP’s of every student computer to be able to connect to them. This is when I reversed the roles and made it so that the teacher’s computer would be the server then all the clients would just need to know 1 address for the teacher’s computer to be able to connect to them. The same information would still be sent over the sockets, but it would just enable the deployment of the application to be a lot more simple and easy.

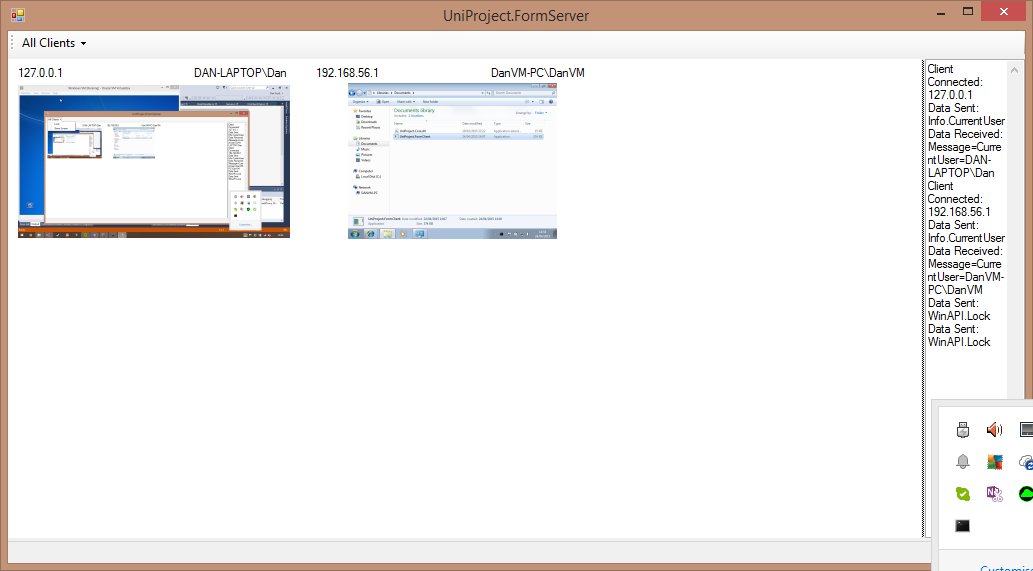
Development then took the turn of being able to send images over the sockets to enable a live screen feed of each client connected to the server as seen here:



Once I had reached the point at which the server architecture was robust and seemed as though most of the error catching and handling had been implemented, I started to work on the user experience end of my application. As this application was designed for a teacher to be able to control student computers in an IT Classroom, I felt that the client end of this application should be hidden and un-closable, so the only user experience from a client point of view is a tray icon. This try icon also displays error messages if the connection to the server is dropped and then picked up again. From a server point of view, I completely re-designed the form to display the computer screen feed as well as the IP of the machine.

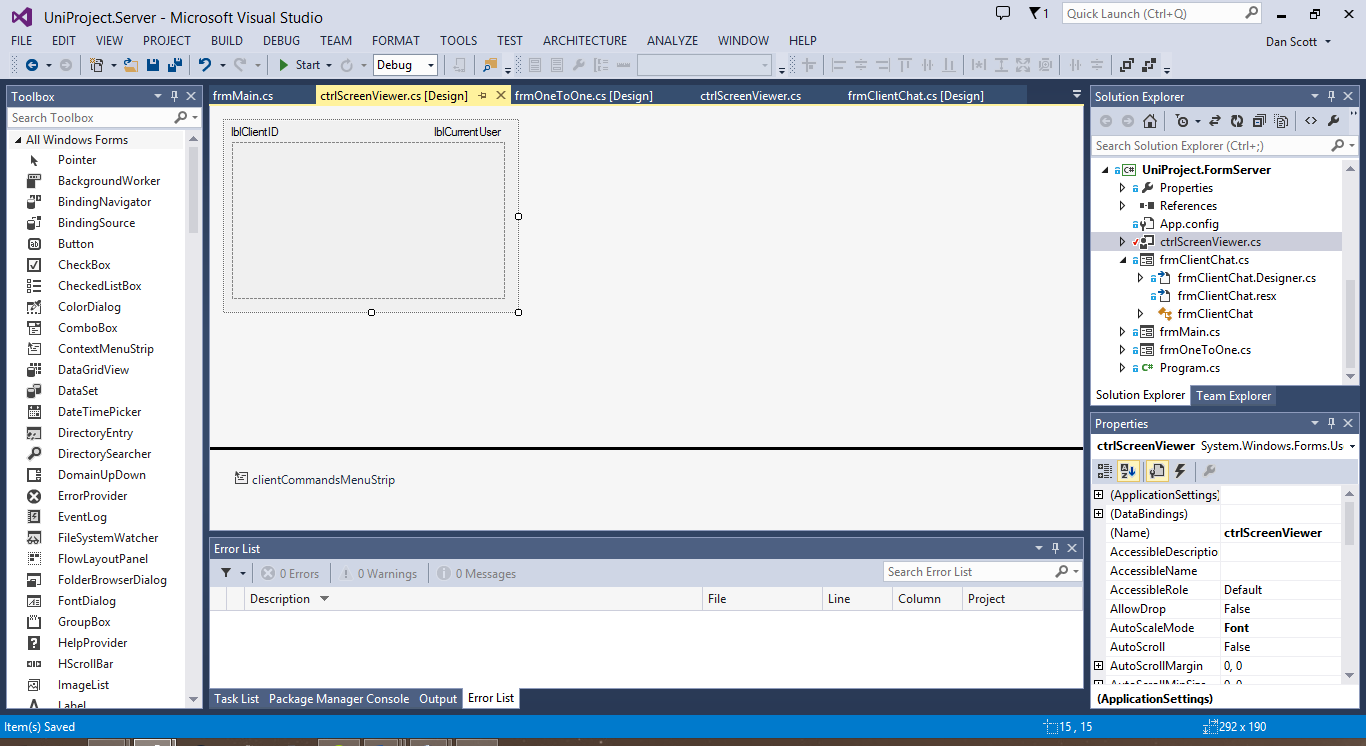
To be able to test the server architecture I also utilised VMWare software to emulate Windows environments and installed the .NET framework on each instance of Windows so that my applications would work within the environments. I later tested this on my friend’s laptops over the same network and it worked as intended. The only features I have implemented at the time of writing this report is an unlimited amount of client connections to the server, locking all the clients that are connected to the server, locking a selected client within the client viewer, viewing a client’s screen full screen and handling if the server drops and then tries to reconnect after 10 seconds.





# Design

The design of my application wasn’t thought through much before development, it was more of a as-you-go-along approach. As I needed more components for my application, I added them at the implementation stage. One of the most useful components I used was one that I created myself as a User Control. This User Control holds a PictureBox for the screen of the client and 2 Label Controls, one for the IP Address of the client and one for the Username of the current user on the machine.



I also implemented a “OneToOne” form. The idea behind this form being that I would be able to have the client screen up full screen on my laptop, to be able to see what the student is doing I greater detail, rather than a little thumbnail.

# Review

## What have I learnt from this project?

Before I started this project, as I have already stated I knew C# but I didn’t have any prior knowledge of socket/server programming so that is something I picked up during the creation of this project. I also had the opportunity to play with the Windows API, which was interesting. Threading was also a part of the .NET framework that I had not explored before and discovering what that was capable of doing was eye opening.

## What I did well

I feel like the server architecture I have used is robust to the point where it’s almost unbreakable. The clients attempt to reconnect if they lost connection to the server. I also feel that the overall look and feel of the application is good enough for any end user to be able to use.

## What could I have added if I had more time

If I had more time I would implement more client side features such as keyboard and mouse input using the windows API as well as launching my own applications from across the network. Another deliverable I didn’t quite manage to achieve was having usernames and passwords of the teachers to enable me to limit the amount of people that can access my application, as well as displaying images full screen on the client side. FTP would also be a nice added feature to implement into my project.

There’s also a question of efficiency especially when sending such large image files over the network. So given more time I would do more research into the best and most practical way of sending and receiving images over the network to reduce the impact as much as I can. I feel like I should be sending the images reduced in size if the teacher isn’t trying to view the client in full screen.

## Project management

I feel that I was on time with the final build of my application. I already knew C# because of an apprenticeship I was on before coming to University so learning a new language wasn’t a factor for me, however I did have to learn new aspects of the C# .NET language to enable to me to do what I wanted to do. I started researching and playing around with code and demos from January of this year and got most of the core programming done by the end of February. Most of my time after that was spent tinkering and bolting on extra features.

# References

MSDN Synchronous server/client socket - <https://msdn.microsoft.com/en-us/library/6y0e13d3%28v=vs.110%29.aspx>   
Synchronous server/client project inspiration - <http://www.addipop.com/addiServ.php> (MMORPG Server emulator written in C#)