Server Component of the Chat Room Lab

This lab is designed to show students how they may can do socket programming in a step by step process in which they switch back and forth from the server to the client in an effort to validate and test what they have just coded.

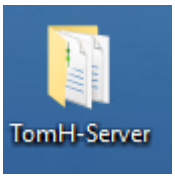
This is the server component; there is also a client component. There are at least three different ways that you could go about the C# network programming; this is just one.

There is very little original programming in this lab; my students are going to design a network game as their final project this semester. They should use many of the principles illustrated in this lab ➔ including emphasis on sound software engineering principles.

Server Part I

1] Download **Chat-Server-Student.zip**

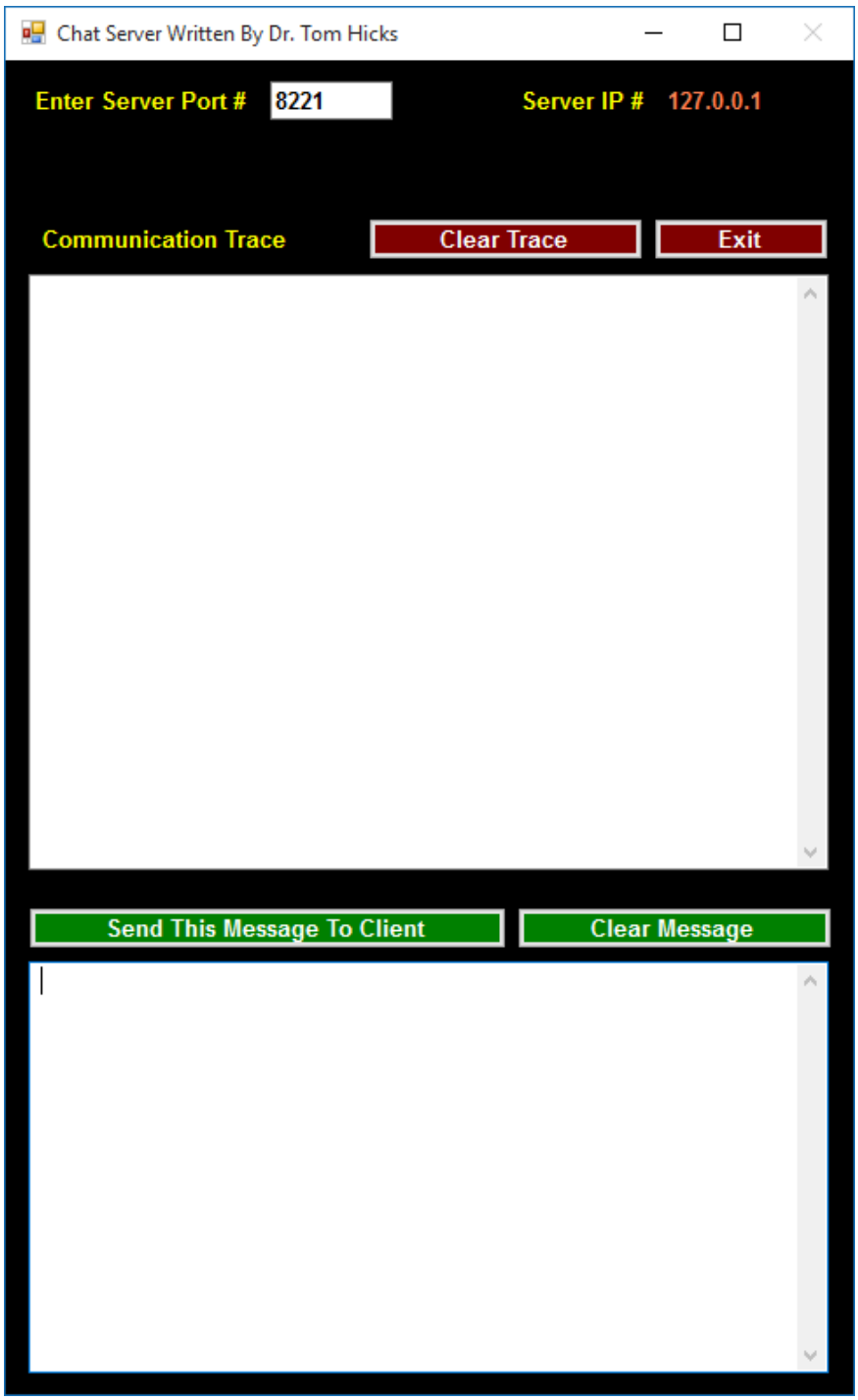
2] Rename the project folder ➔ First Name + Last Initial + Dash + "-Server"



3] Inside this folder you will have the following:

Name	Date modified	Type	Size
bin	10/9/2015 9:00 AM	File folder	
obj	10/9/2015 9:00 AM	File folder	
Properties	10/9/2015 9:00 AM	File folder	
App.config	10/3/2015 3:24 PM	XML Configuratio...	1 KB
Chat_Server.csproj	10/3/2015 3:41 PM	Visual C# Project f...	4 KB
Chat_Server.sln	10/3/2015 3:24 PM	Expression Blend ...	1 KB
Program.cs	10/3/2015 3:41 PM	Visual C# Source f...	1 KB
Server.cs	10/7/2015 11:24 AM	Visual C# Source f...	5 KB
Server.Designer.cs	10/7/2015 5:59 AM	Visual C# Source f...	12 KB
Server.resx	10/7/2015 5:59 AM	.NET Managed Re...	6 KB

4] In order to save my students design time, I have created a basic Chat Server form for them to use.



4] As my students go through this lab, I will ask them to do Server (Part 1) ➔ then Client (Part 1) ➔ Server (Part 2) ➔ then Client (Part 2) etc.

#1 Add Components

1] Add the following components.

```
Chat_Server                                   Chat_Server.Server
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

////////////////////////////////////////////////////////////////////////////
//                                Add Components                          //
////////////////////////////////////////////////////////////////////////////

using System.IO;
using System.Net.Sockets;
using System.Threading;
```

2] Compile!

#2 Set Up Thread

1] Declare the thread;

```
public partial class Server : Form
{
    Thread ReadClientMessage;
```

2] Set up and start the thread;

```
public Server()
{
    InitializeComponent();
    ThreadStart ts = new ThreadStart(RunChatServer);

    ReadClientMessage = new Thread(ts);
    ReadClientMessage.IsBackground = true;
    ReadClientMessage.Start();
}
```

3] **Will Not Compile!**

4] Add functions for **ReceiveMessage**

```
public void RunChatServer()
{
}
```

5] Will Compile!

#3 Need A Tcplistener → Set iPaddress address to loopback

1] Declare a **TCPListener**

```
public partial class Server : Form
{
    Thread ReadClientMessage;
    TcpListener tcpListener;
```

2] Found on Internet → Goal

```
//  TcpListener tcpListener = new TcpListener(IPAddress address, int port);
```

3] How do I fill **address** with your IP? Do a quick Internet Search!

```
public void RunChatServer()
{
    IPAddress address =

    //  TcpListener tcpListener = new TcpListener(IPAddress address, int port);
}
```

4] I found this on the Internet – Does Not Compile → Any thoughts on what I might be missing?

```
public void RunChatServer()
{
    IPAddress address = IPAddress.Parse("131.194.34.10");

    //  TcpListener tcpListener = new TcpListener(IPAddress address, int port);
}
```

5] That's it. I need **System.Net**

```
////////////////////////////////////////////////////////////////////////
//                              Add Components                        //
////////////////////////////////////////////////////////////////////////

using System.IO;
using System.Net.Sockets;
using System.Threading;
using System.Net;
```

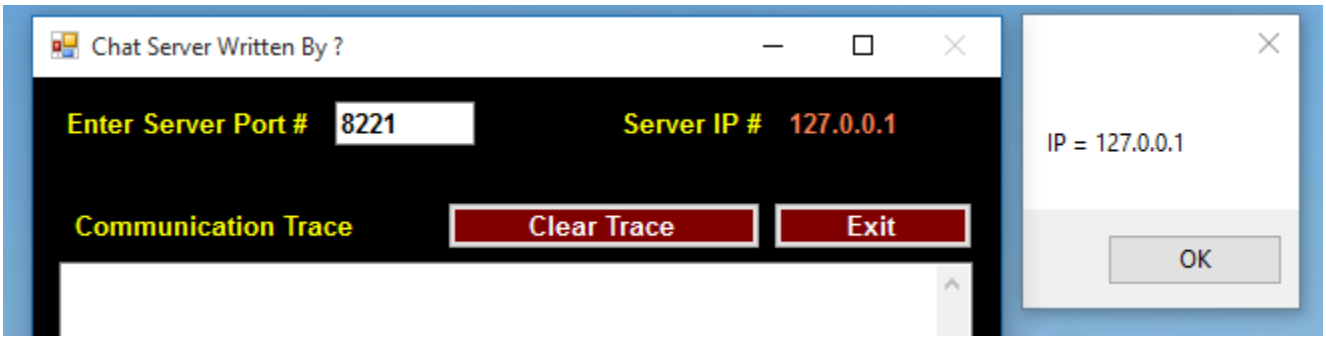
6] This code below compiles.

#4 Need A Tcplistener → Set The IPAddress & Port

1] Use MessageBox to show the value in lbIP.

```
public void RunChatServer()
{
    IPAddress address = IPAddress.Parse("131.194.34.10");

    MessageBox.Show("IP = " + lbIP.Text.ToString());

    //  TcpListener tcpListener = new TcpListener(IPAddress address, int port);
}
```

2] Output:



3] How can we replace the '131;194.34.10" with the value in **lbIP**?

```
public void RunChatServer()
{
//  IPAddress address = IPAddress.Parse("131.194.34.10");
    IPAddress address = IPAddress.Parse(lbIP.Text.ToString());

    MessageBox.Show("IP = " + lbIP.Text.ToString());

    //  TcpListener tcpListener = new TcpListener(IPAddress address, int port);
}
```

4] How get the port number into a variable such as **port**? ➔

```
public void RunChatServer()
{
//  IPAddress address = IPAddress.Parse("131.194.34.10");
    IPAddress address = IPAddress.Parse(lbIP.Text.ToString());
    int port =

    MessageBox.Show("IP = " + lbIP.Text.ToString());

    //  TcpListener tcpListener = new TcpListener(IPAddress address, int port);
}
```

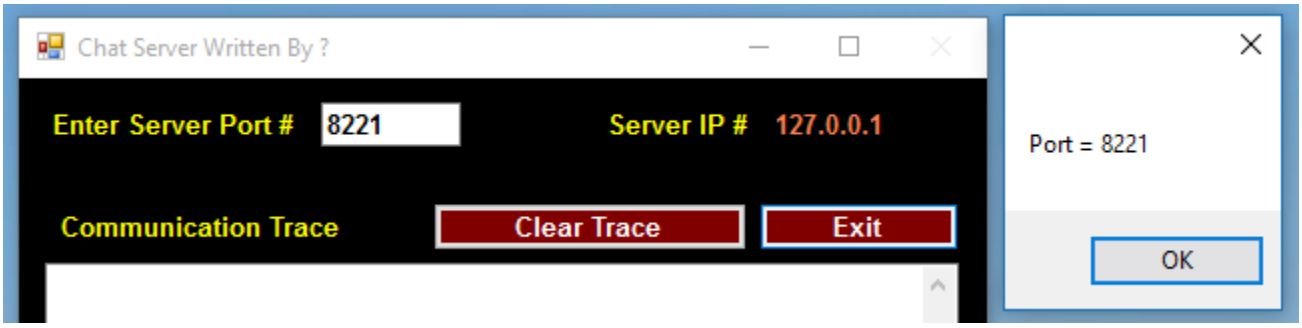
5] How get the port number into a variable such as port? ➔ Enter The following:

```
public void RunChatServer()
{
//  IPAddress address = IPAddress.Parse("131.194.34.10");
    IPAddress address = IPAddress.Parse(lbIP.Text.ToString());
    int port = Convert.ToInt16(txtPortNo.Text);

    MessageBox.Show("IP = " + lbIP.Text.ToString());

    //  TcpListener tcpListener = new TcpListener(IPAddress address, int port);
}
```

6] Use MessageBox to show the value in txtPortNo.

```
public void RunChatServer()
{
//  IPAddress address = IPAddress.Parse("131.194.34.10");
    IPAddress address = IPAddress.Parse(lbIP.Text.ToString());
    int port = Convert.ToInt16(txtPortNo.Text);

    MessageBox.Show("IP = " + lbIP.Text.ToString());
    MessageBox.Show("Port = " + txtPortNo.Text);

    //  TcpListener tcpListener = new TcpListener(IPAddress address, int port);
}
```

7] Output:



#5 Need A Tcplistener → Set The IPAddress & Port

1] We want to create a **TCPListener** ➔ Make the following:

```
public void RunChatServer()
{
//  IPAddress address = IPAddress.Parse("131.194.34.10");
    IPAddress address = IPAddress.Parse(lbIP.Text.ToString());
    int port = Convert.ToInt16(txtPortNo.Text);

    // #1 ==> Create & Start The Listener
    tcpListener = new TcpListener(address, port);
}
```

2] This code below compiles.

3] Because I was doing this is stages, I was unable to terminate the listener properly. Add the following:

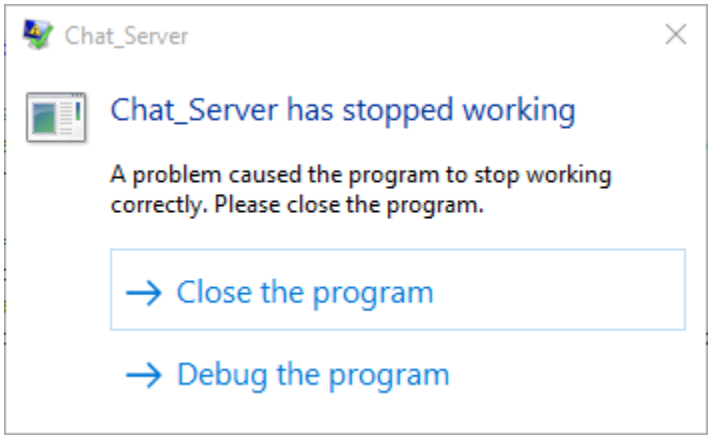
```
public void RunChatServer()
{
//  IPAddress address = IPAddress.Parse("131.194.34.10");
    IPAddress address = IPAddress.Parse(lbIP.Text.ToString());
    int port = Convert.ToInt16(txtPortNo.Text);

    // #1 ==> Create & Start The Listener
    tcpListener = new TcpListener(address, port);
    // Because I had socket reuse errors during step by step creation
    tcpListener.Server.SetSocketOption(SocketOptionLevel.Socket, SocketOptionName.ReuseAddress, true);
}
```

4] This code below compiles.

5] Change to an invalid address.

```
public void RunChatServer()
{
  IPAddress address = IPAddress.Parse("131.194.34.1055");
//  IPAddress address = IPAddress.Parse(lbIP.Text.ToString());
    int port = Convert.ToInt16(txtPortNo.Text);

    // #1 ==> Create & Start The Listener
    tcpListener = new TcpListener(address, port);
    // Because I had socket reuse errors during step by step creation
    tcpListener.Server.SetSocketOption(SocketOptionLevel.Socket, SocketOptionName.ReuseAddress, true);
}
```

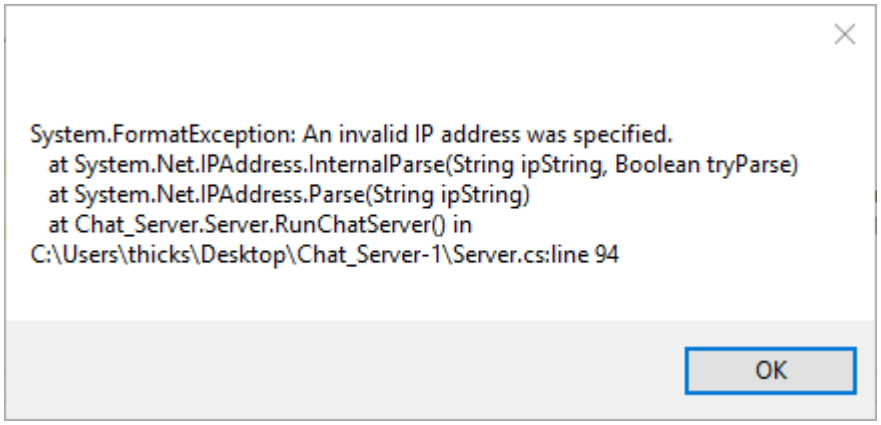
6] This code below compiles but execution leaves much to be desired.



7] Add A Try-Catch to the listener.

```
public void RunChatServer()
{
    int port = Convert.ToInt16(txtPortNo.Text);
    try
    {
        IPAddress address = IPAddress.Parse("131.194.34.1055");
        //  IPAddress address = IPAddress.Parse(lbIP.Text.ToString());

        // #1 ==> Create & Start The Listener
        tcpListener = new TcpListener(address, port);
        // Because I had socket reuse errors during step by step creation
        tcpListener.Server.SetSocketOption(SocketOptionLevel.Socket, SocketOptionName.ReuseAddress, true);
    }

    catch (Exception error)
    {
        MessageBox.Show(error.ToString());
    }
}
```

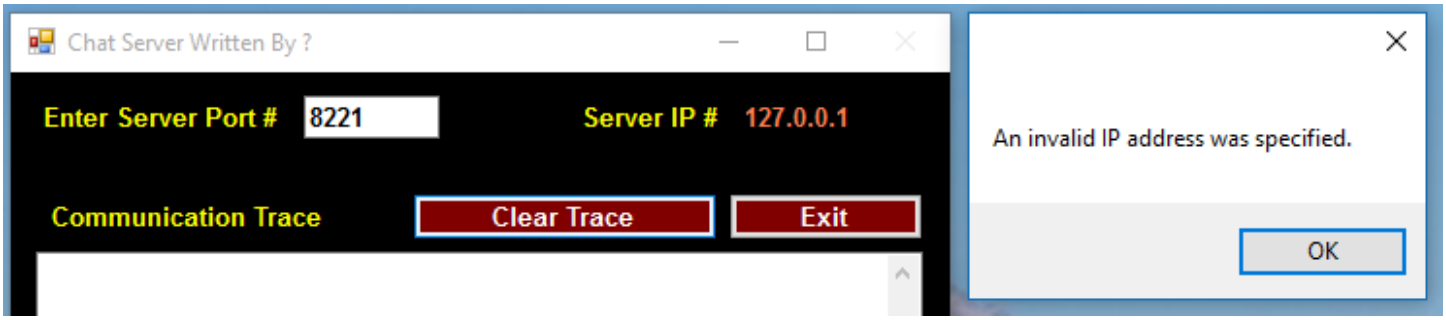
8] We can dig the error message out, but this would not be good for users.



9] Modify the Try-Catch!

```
try
{
    IPAddress address = IPAddress.Parse("131.194.34.1055");
    //  IPAddress address = IPAddress.Parse(lbIP.Text.ToString());

    // #1 ==> Create & Start The Listener
    tcpListener = new TcpListener(address, port);
    // Because I had socket reuse errors during step by step creation
    tcpListener.Server.SetSocketOption(SocketOptionLevel.Socket, SocketOptionName.ReuseAddress, true);
}

catch (Exception error)
{
    MessageBox.Show(error.Message.ToString());
}
```

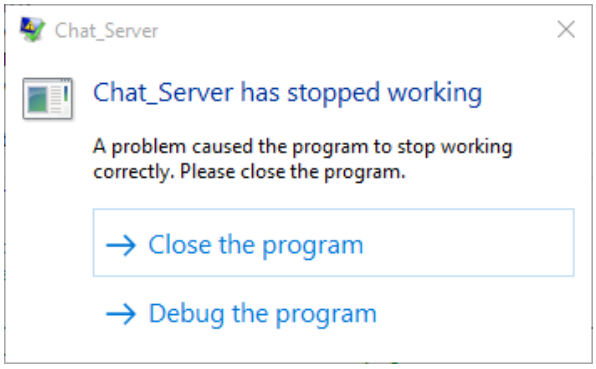
10] Much better output.



11] Make the following change. Port No is a user input & they will make errors.

```
        public void RunChatServer()
        {
//          int port = Convert.ToInt16(txtPortNo.Text);
            int port = Convert.ToInt16("999999");
            try
            {
                IPAddress address = IPAddress.Parse("131.194.34.1055");
                //  IPAddress address = IPAddress.Parse(lbIP.Text.ToString());

                // #1 ==> Create & Start The Listener
                tcpListener = new TcpListener(address, port);
                // Because I had socket reuse errors during step by step creation
                tcpListener.Server.SetSocketOption(SocketOptionLevel.Socket, SocketOptionName.ReuseAddress, true);
            }

            catch (Exception error)
            {
                MessageBox.Show(error.Message.ToString());
            }
        }
```

12] Once again we get the very meaningful error message: [Suggestions?]



13] Sure ➔ Put it inside try-catch:

```
public void RunChatServer()
{
    try
    {
        //  IPAddress address = IPAddress.Parse("131.194.34.10");
        IPAddress address = IPAddress.Parse(lbIP.Text.ToString());
        //  int port = Convert.ToInt16(txtPortNo.Text);
        int port = Convert.ToInt16("999999");

        // #1 ==> Create & Start The Listener
        tcpListener = new TcpListener(address, port);
        // Because I had socket reuse errors during step by step creation
        tcpListener.Server.SetSocketOption(SocketOptionLevel.Socket, SocketOptionName.ReuseAddress, true);
    }

    catch (Exception error)
    {
        MessageBox.Show(error.Message.ToString());
    }
}
```

14] Output:

15] Definitely an invalid Port No. But What about 65000 ➔ This is a problem

15] This works.

```
public void RunChatServer()
{
    try
    {
        //  IPAddress address = IPAddress.Parse("131.194.34.10");
        IPAddress address = IPAddress.Parse(lbIP.Text.ToString());
        //  int port = Convert.ToInt32(txtPortNo.Text);
        int port = Convert.ToInt32("65000");

        // #1 ==> Create & Start The Listener
        tcpListener = new TcpListener(address, port);
        // Because I had socket reuse errors during step by step creation
        tcpListener.Server.SetSocketOption(SocketOptionLevel.Socket, SocketOptionName.ReuseAddress, true);
    }

    catch (Exception error)
    {
        MessageBox.Show(error.Message.ToString());
    }
}
```

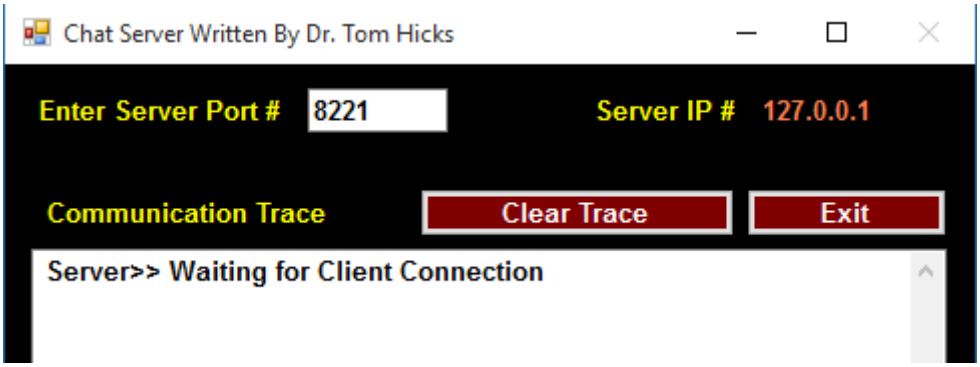
#6 Start the Listener & Send A Status Update To The Communication Trace

1] Start the Listener. Send the update!

```
public void RunChatServer()
{
    try
    {
        //  IPAddress address = IPAddress.Parse("131.194.34.10");
        IPAddress address = IPAddress.Parse(lbIP.Text.ToString());
        //  int port = Convert.ToInt32(txtPortNo.Text);
        int port = Convert.ToInt32("65000");

        // #1 ==> Create & Start The Listener
        tcpListener = new TcpListener(address, port);
        // Because I had socket reuse errors during step by step creation
        tcpListener.Server.SetSocketOption(SocketOptionLevel.Socket, SocketOptionName.ReuseAddress, true);


        // #2 ==> Listener Begins Waiting For Connection
        tcpListener.Start();

        txtCommunicationTrace.Text += "Server>> " + "Waiting for Client Connection\r\n";
    }

    catch (Exception error)
    {
        MessageBox.Show(error.Message.ToString());
    }
}
```

2] Compiles

3] Output:



#7 Create the Client Socket Connection & Update To The Communication Trace

1] Change the port.

2] Declare the Socket

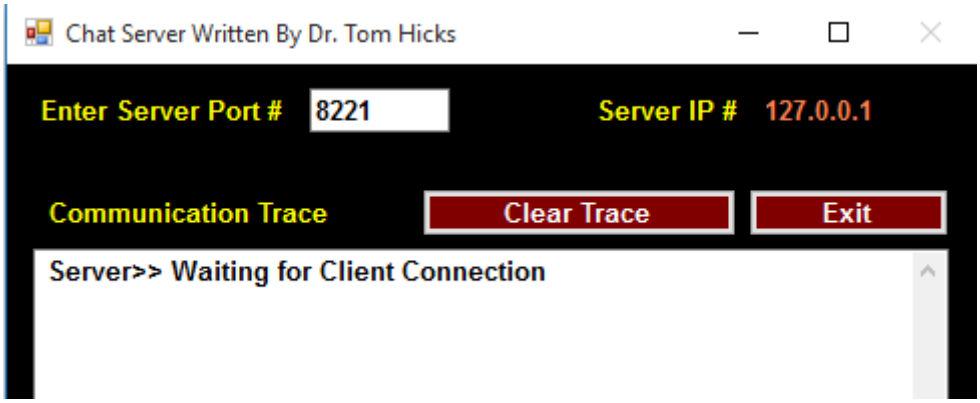
```
public partial class Server : Form
{
    Thread ReadClientMessage;
    TcpListener tcpListener;
    Socket socketConnection;
```

3] Create the client connection & update the trace.

```
public void RunChatServer()
{
    try
    {
        //  IPAddress address = IPAddress.Parse("131.194.34.10");
        IPAddress address = IPAddress.Parse(lbIP.Text.ToString());
        int port = Convert.ToInt32(txtPortNo.Text);
        //  int port = Convert.ToInt32("65000");

        // #1 ==> Create & Start The Listener
        tcpListener = new TcpListener(address, port);
        // Because I had socket reuse errors during step by step creation
        tcpListener.Server.SetSocketOption(SocketOptionLevel.Socket, SocketOptionName.ReuseAddress, true);


        // #2 ==> Listener Begins Waiting For Connection
        tcpListener.Start();

        txtCommunicationTrace.Text += "Server>> " + "Waiting for Client Connection\r\n";

        // #3 ==> Create the Client Connection
        socketConnection = tcpListener.AcceptSocket();

        txtCommunicationTrace.Text += "Server>> " + "Client Connection Complete";
    }

    catch (Exception error)
    {
        MessageBox.Show(error.Message.ToString());
    }
}
```

4] This does compile.

5] Note that it really is waiting. We know that because we do not see the **Client Connection Complete** message.



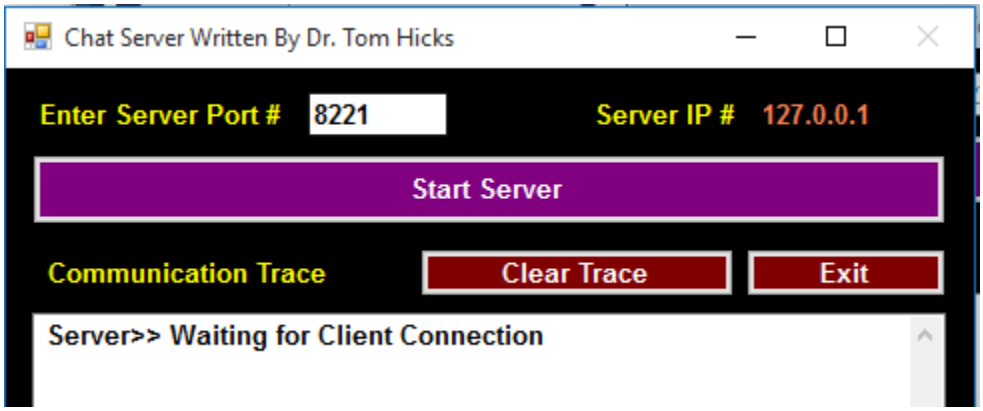
Do Part I On Client

Server Part II

1] **Making some progress, but we have a MAJOR DESIGN FLAW in our processing that is so obvious that you should be able to tell me what it is?**

#7 No Opportunity To Change Port

1] Create button **btnStartListening**



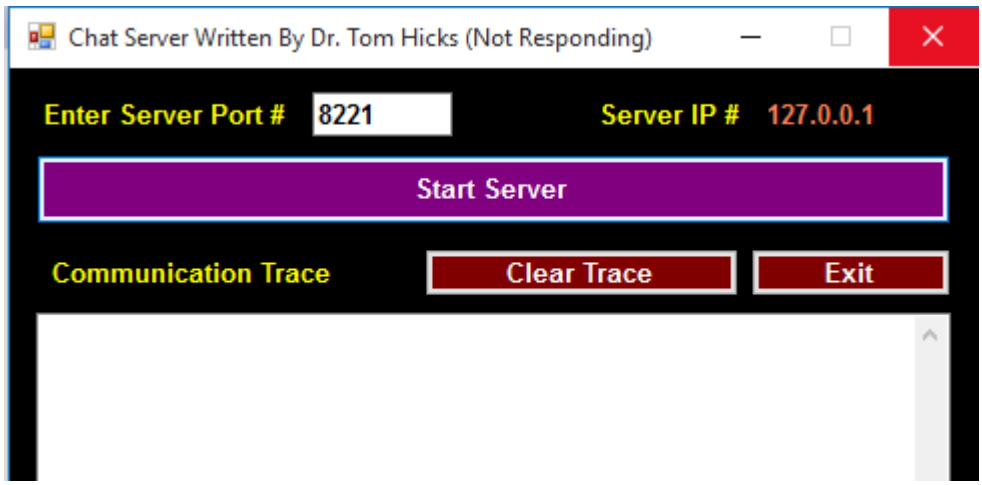
2] What code do you suppose we ought to move to this button?

```
private void btnStartServer_Click(object sender, EventArgs e)
{
    try
    {
        //  IPAddress address = IPAddress.Parse("131.194.34.10");
        IPAddress address = IPAddress.Parse(lbIP.Text.ToString());
        int port = Convert.ToInt32(txtPortNo.Text);
        //  int port = Convert.ToInt32("65000");

        // #1 ==> Create & Start The Listener
        tcpListener = new TcpListener(address, port);
        // Because I had socket reuse errors during step by step creation
        tcpListener.Server.SetSocketOption(SocketOptionLevel.Socket, SocketOptionName.ReuseAddress, true);


        // #2 ==> Listener Begins Waiting For Connection
        tcpListener.Start();

        txtCommunicationTrace.Text += "Server>> " + "Waiting for Client Connection\r\n";

        // #3 ==> Create the Client Connection
        socketConnection = tcpListener.AcceptSocket();

        txtCommunicationTrace.Text += "Server>> " + "Client Connection Complete";
    }

    catch (Exception error)
    {
        MessageBox.Show(error.Message.ToString());
    }
}
```

3] What code is left in RunChatServer?

```
public void RunChatServer()
{

}
```

4] Start the Chat Server



5] I wanted to see a message saying that we are **Waiting for Client Connection** ; why don't we see it? Fix this.

6] Notice that even if you put the waiting at the top of the button code, you do not see the message. Any thoughts?

```
private void btnStartServer_Click(object sender, EventArgs e)
{
    txtCommunicationTrace.Text += "Server>> " + "Waiting for Client Connection\r\n";
    try
    {

        //  IPAddress address = IPAddress.Parse("131.194.34.10");
        IPAddress address = IPAddress.Parse(lbIP.Text.ToString());
        int port = Convert.ToInt32(txtPortNo.Text);

        // #1 ==> Create & Start The Listener
        tcpListener = new TcpListener(address, port);
        // Because I had socket reuse errors during step by step creation
        tcpListener.Server.SetSocketOption(SocketOptionLevel.Socket, SocketOptionName.ReuseAddress, true);

        //  #2 ==> Listener Begins Waiting For Connection
        tcpListener.Start();


        //  #3 ==> Create the Client Connection
        socketConnection = tcpListener.AcceptSocket();

        txtCommunicationTrace.Text += "Server>> " + "Client Connection Complete";
    }

    catch (Exception error)
    {
        MessageBox.Show(error.Message.ToString());
    }
}
```

7] What happens if you push the Start Listening button twice? – PROBLEMS – this code is no longer in the thread.

8] Return the code to RunChatServer → How do we get it to execute only after the user pushes the Start Server button?

9] May be many solutions, but this might point you in the right direction? Declare ServerStartInitiated.

```
public partial class Server : Form
{
    Thread ReadClientMessage;
    TcpListener tcpListener;
    Socket socketConnection;
    bool ServerStartInitiated = false;
```

10] Set the variable to true when the button is pushed.

```
private void btnStartServer_Click(object sender, EventArgs e)
{
    ServerStartInitiated = true;
}
```

11] My thought is to do something like this. What do you think?

```
public void RunChatServer()
{

    if (ServerStartInitiated)
    {
        txtCommunicationTrace.Text += "Server>> " + "Waiting for Client Connection\r\n";
        try
        {

            //  IPAddress address = IPAddress.Parse("131.194.34.10");
            IPAddress address = IPAddress.Parse(lbIP.Text.ToString());
            int port = Convert.ToInt32(txtPortNo.Text);

            // #1 ==> Create & Start The Listener
            tcpListener = new TcpListener(address, port);
            // Because I had socket reuse errors during step by step creation
            tcpListener.Server.SetSocketOption(SocketOptionLevel.Socket, SocketOptionName.ReuseAddress, true);

            //  #2 ==> Listener Begins Waiting For Connection
            tcpListener.Start();


            //  #3 ==> Create the Client Connection
            socketConnection = tcpListener.AcceptSocket();

            txtCommunicationTrace.Text += "Server>> " + "Client Connection Complete";
        }

        catch (Exception error)
        {
            MessageBox.Show(error.Message.ToString());
        }
    }

}
```

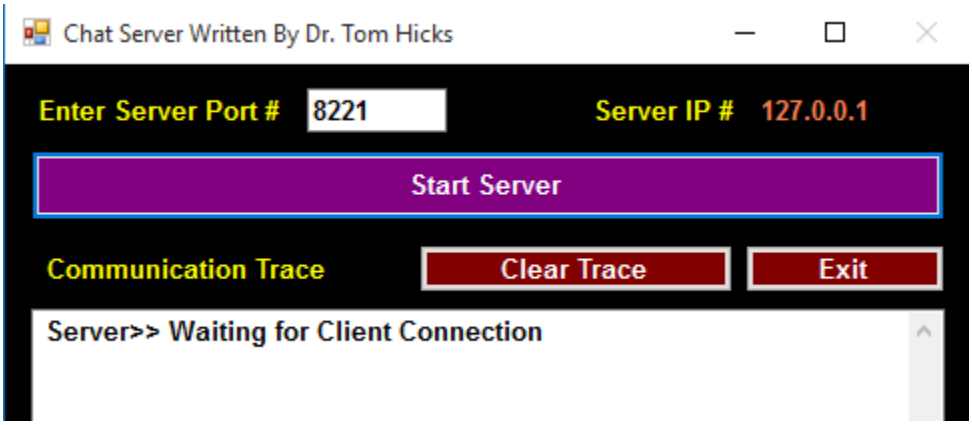
12] Run the code. The if-block is never executed. Have we prevented the crash caused by the user pushing the connect button twice? DON’T KNOW SINCE NOT LISTENING.

13] The RunChatServer needs an event loop. It starts and is done.

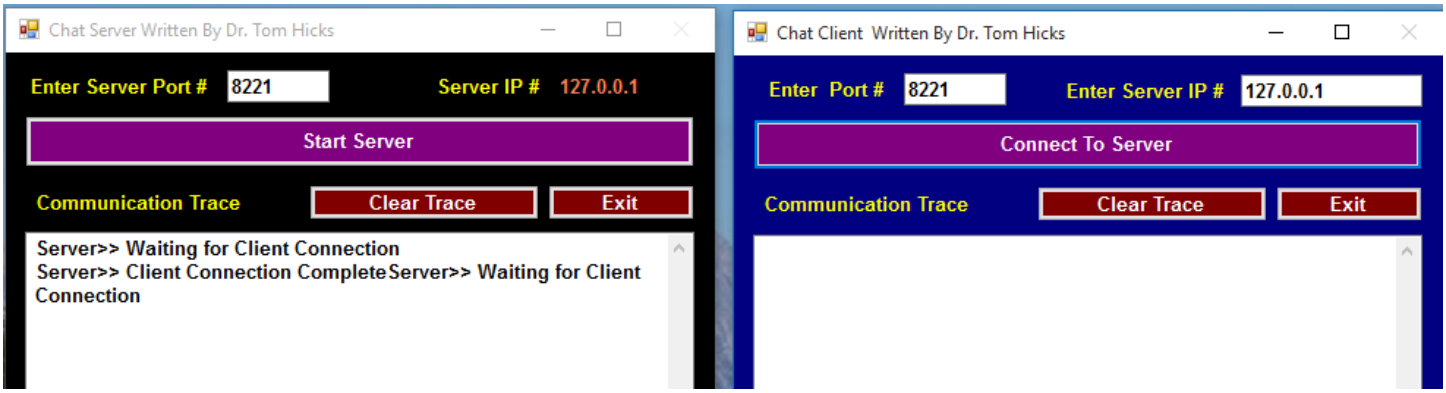
14] My first shot at an event loop..

```
public void RunChatServer()
{
    while (true)
    {
        if (ServerStartInitiated)
        {
            txtCommunicationTrace.Text += "Server>> " + "Waiting for Client Connection\r\n";
            try
            {

                //  IPAddress address = IPAddress.Parse("131.194.34.10");
                IPAddress address = IPAddress.Parse(lbIP.Text.ToString());
                int port = Convert.ToInt32(txtPortNo.Text);

                // #1 ==> Create & Start The Listener
                tcpListener = new TcpListener(address, port);
                // Because I had socket reuse errors during step by step creation
                tcpListener.Server.SetSocketOption(SocketOptionLevel.Socket, SocketOptionName.ReuseAddress, true);

                //  #2 ==> Listener Begins Waiting For Connection
                tcpListener.Start();


                //  #3 ==> Create the Client Connection
                socketConnection = tcpListener.AcceptSocket();

                txtCommunicationTrace.Text += "Server>> " + "Client Connection Complete";
            }

            catch (Exception error)
            {
                MessageBox.Show(error.Message.ToString());
            }
        }
    }

}
```

15] I start the server and all looks good.



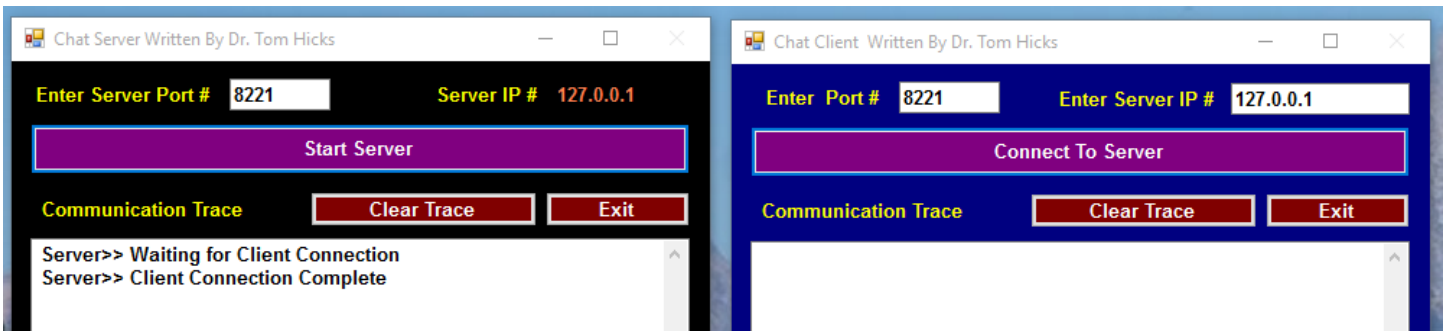
15] I Connect to Server from the client and see that it is setting up the listener again! BAD BAD! We only want the program to do this once. FIX THE PROBLEM!



16] One solution might be:

```
        public void RunChatServer()
        {
            while (true)
            {
                //==========================================================================
                //==                Establish The Client Socket Connection                ==
                //==========================================================================
                if (ServerStartInitiated)
                {
                    ServerStartInitiated = false;
                    try
                    {

                        //  IPAddress address = IPAddress.Parse("131.194.34.10");
                        IPAddress address = IPAddress.Parse(lbIP.Text.ToString());
                        int port = Convert.ToInt32(txtPortNo.Text);

                        // #1 ==> Create & Start The Listener
                        tcpListener = new TcpListener(address, port);
                        // Because I had socket reuse errors during step by step creation
                        tcpListener.Server.SetSocketOption(SocketOptionLevel.Socket, SocketOptionName.ReuseAddress, true);

                        //  #2 ==> Listener Begins Waiting For Connection
                        tcpListener.Start();

                        txtCommunicationTrace.Text += "Server>> " + "Waiting for Client Connection\r\n";

                        //  #3 ==> Create the Client Connection
                        socketConnection = tcpListener.AcceptSocket();

                        txtCommunicationTrace.Text += "Server>> " + "Client Connection Complete\r\n";
                    }

                    catch (Exception error)
                    {
                        MessageBox.Show(error.Message.ToString());
                    }
                }
            }

        }
```

17] Output:


Chat Server Written By Dr. Tom Hicks
Enter Server Port #
8221
Server IP #
127.0.0.1
Start Server
Communication Trace
Clear Trace
Exit
Server>> Waiting for Client Connection
Server>> Client Connection Complete
Chat Client Written By Dr. Tom Hicks
Enter Port #
8221
Enter Server IP #
127.0.0.1
Connect To Server
Communication Trace
Clear Trace
Exit

#8 Work On The User Interface

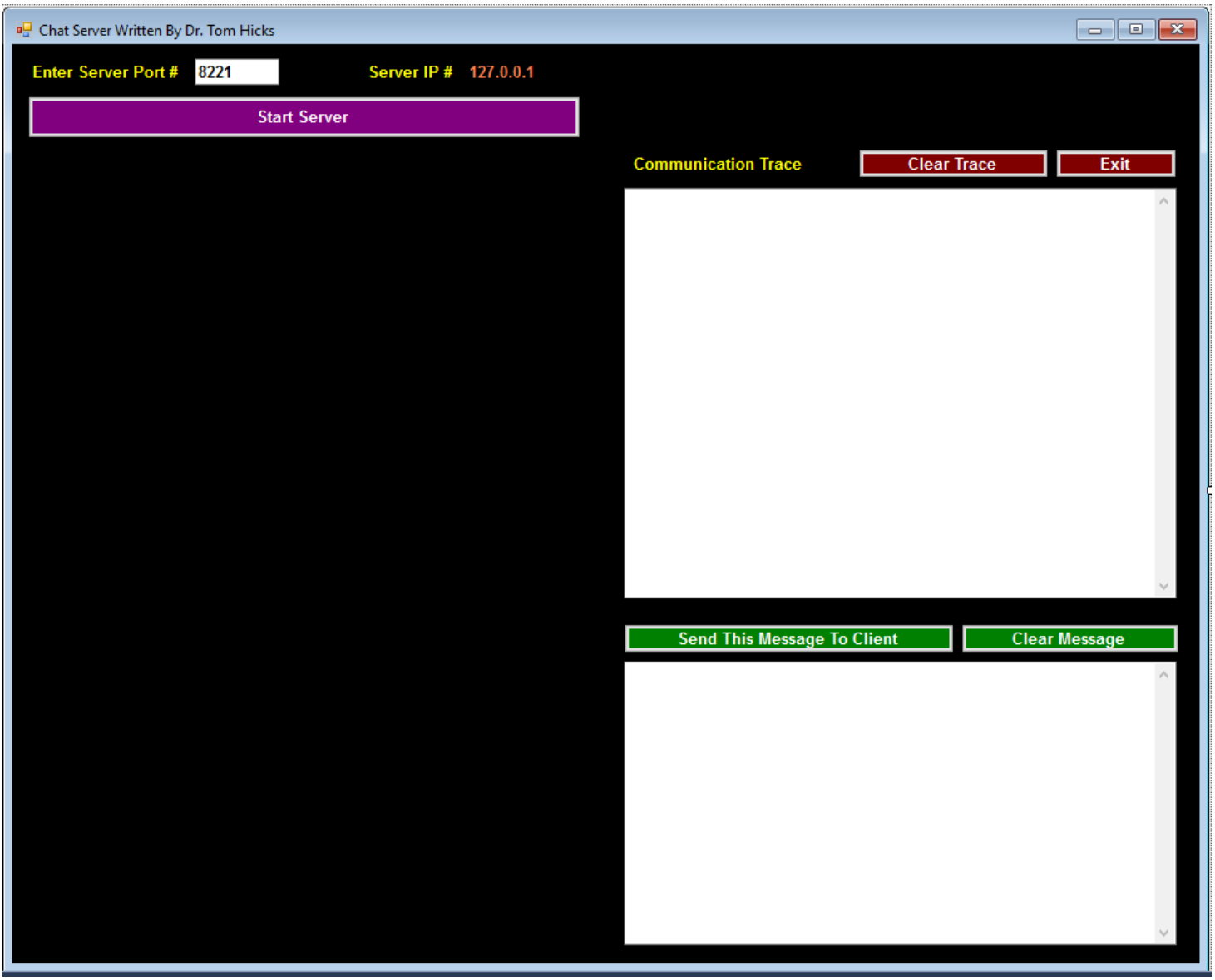
1] Can I push the Start Server button twice? YES! How prevent?

2] What is the first thing the user ought to do on the server? SET PORT & START

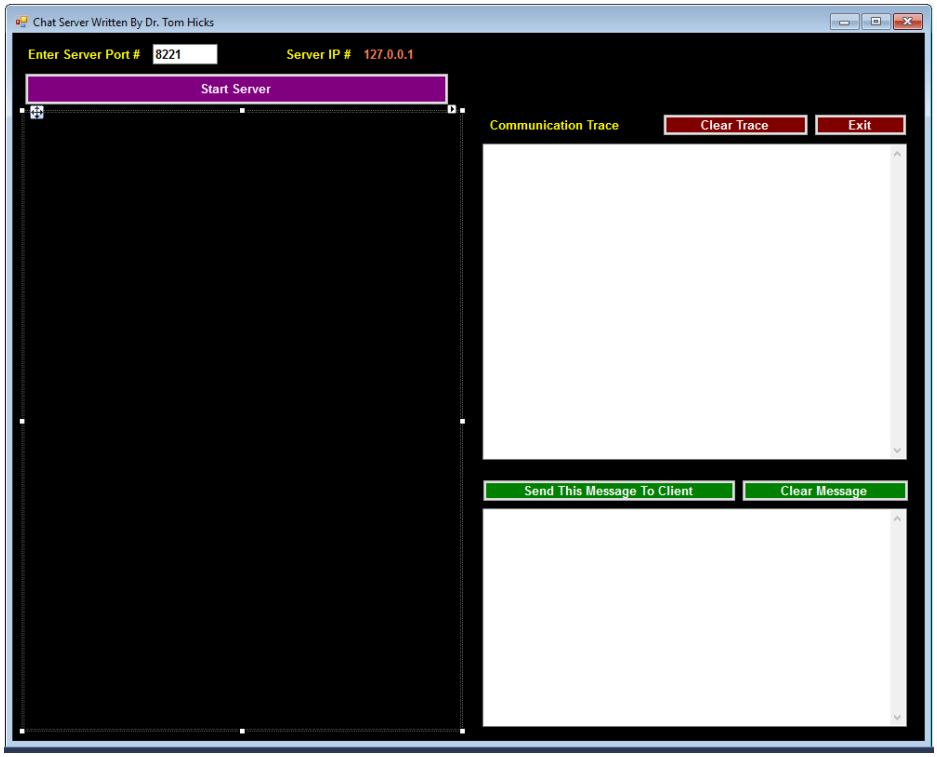
3] There are other buttons and fields that can distract from this task. Any thoughts on how we could force them into doing what is right without a user manual or instructions?

4] There are a small number of items on this form. We could hide/disable them one at a time, but this will not be a good solution for your final project.

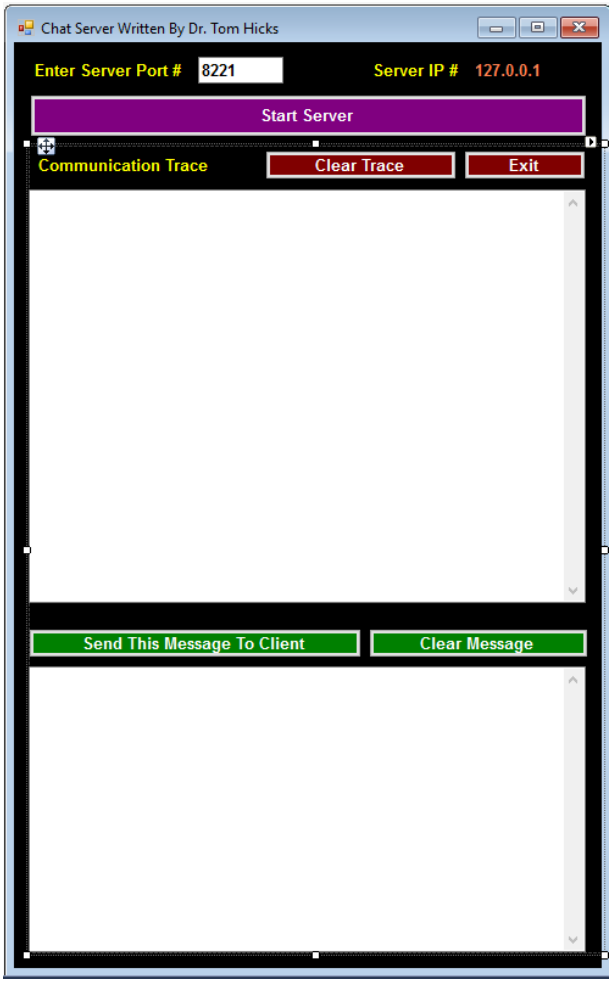
5] Maybe drag these controls to one side.



6] Add a Panel



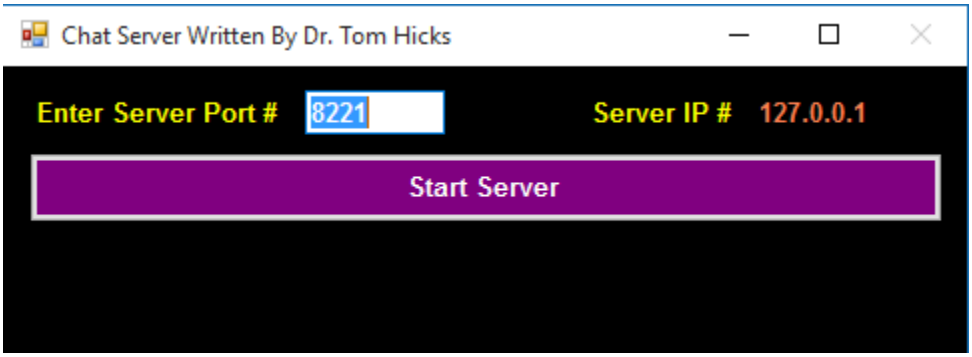
7] Drag the items on the right onto the panel.



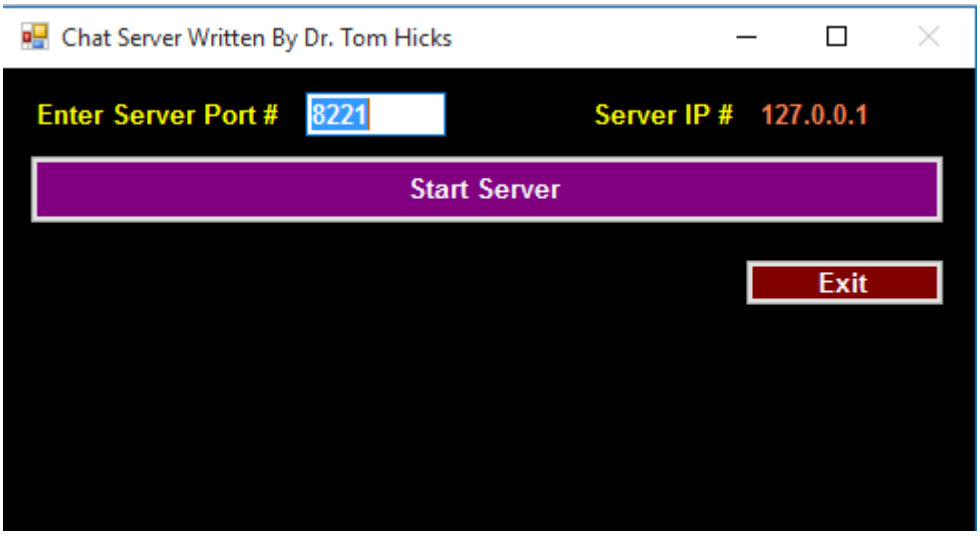
8] Hide the panel in the page load.

```
private void Server_Load(object sender, EventArgs e)
{
    panel1.Hide();
}
```

9] Better, but I am missing something? THOUGHTS? WHAT IF THE USER WANTS TO EXIT?



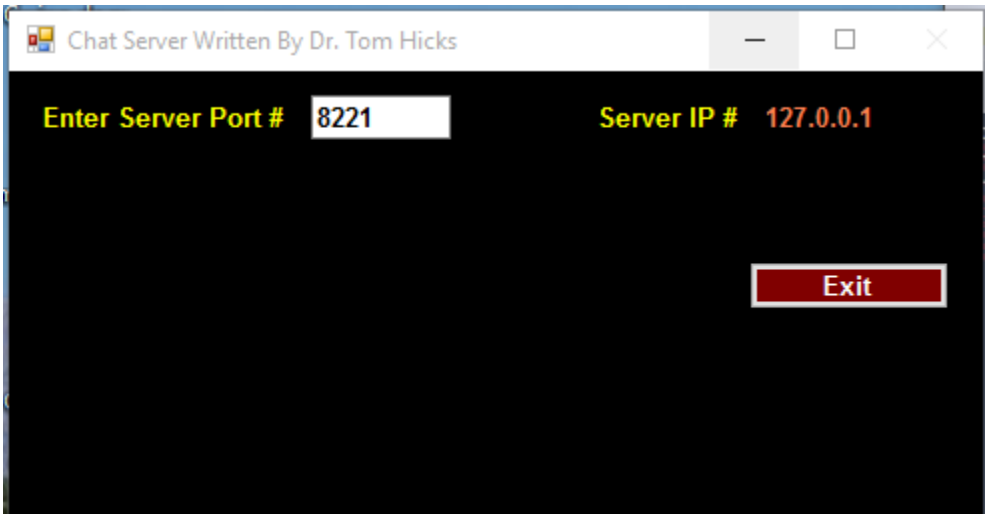
10] Move the Exit button off the panel. My choices → Change Port, Start Server, or Exit.



11] What can we do to prevent starting the server multiple times? HIDE THE START SERVER button. Where do we do that?

```
try
{
    //  IPAddress address = IPAddress.Parse("131.194.34.10");
    IPAddress address = IPAddress.Parse(lbIP.Text.ToString());
    int port = Convert.ToInt32(txtPortNo.Text);

    // #1 ==> Create & Start The Listener
    tcpListener = new TcpListener(address, port);
    // Because I had socket reuse errors during step by step creation
    tcpListener.Server.SetSocketOption(SocketOptionLevel.Socket, SocketOptionName.ReuseAddress, true);

    //  #2 ==> Listener Begins Waiting For Connection
    tcpListener.Start();

    txtCommunicationTrace.Text += "Server>> " + "Waiting for Client Connection\r\n";
    btnStartServer.Hide();
    //  #3 ==> Create the Client Connection
    socketConnection = tcpListener.AcceptSocket();

    txtCommunicationTrace.Text += "Server>> " + "Client Connection Complete\r\n";
}
```

12] But this leaves us in a positions where the user can't see our message about Waiting for Client Connection. Move txtCommunicationTrace off the panel and manage it separately.



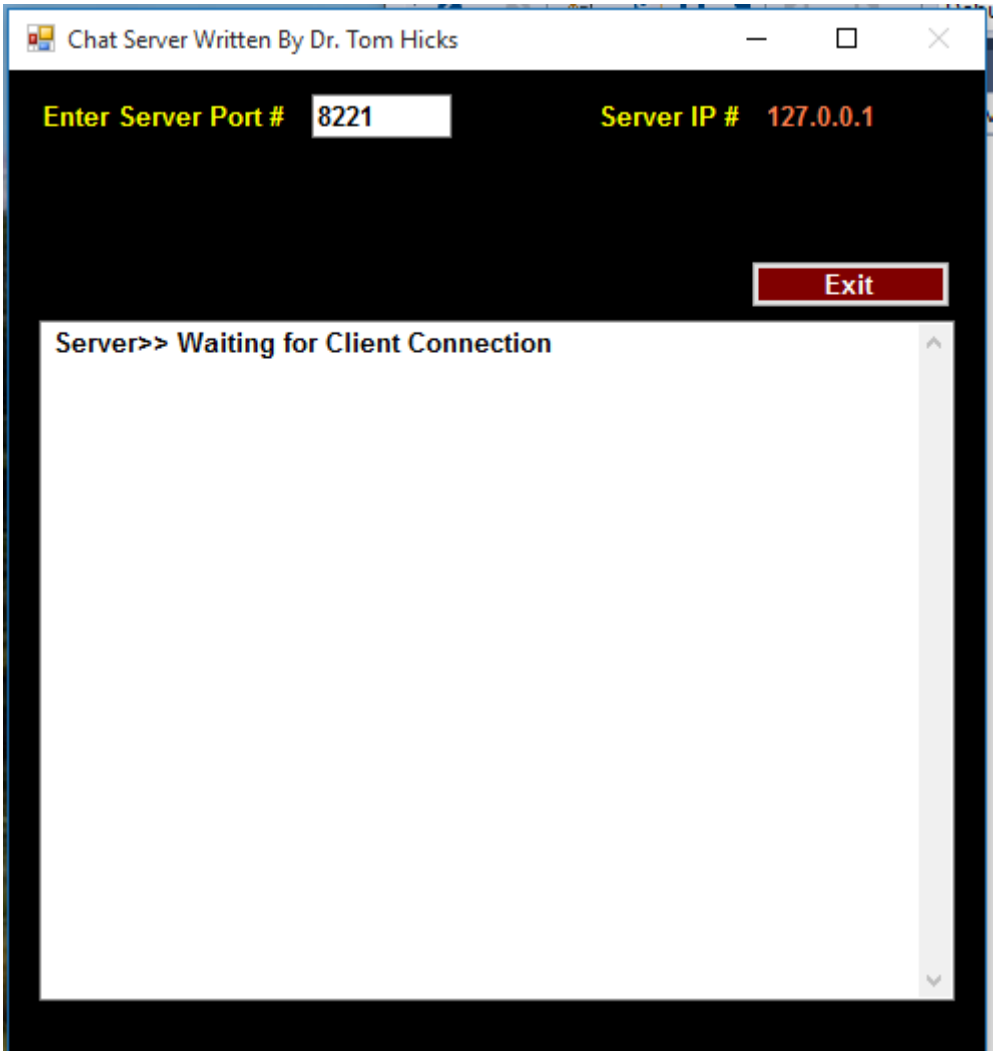
13] Hide it in the load.

```
private void Server_Load(object sender, EventArgs e)
{
    panel1.Hide();
    txtCommunicationTrace.Hide();
}
```

14] Show it when waiting.

```
try
{
    //  IPAddress address = IPAddress.Parse("131.194.34.10");
    IPAddress address = IPAddress.Parse(lbIP.Text.ToString());
    int port = Convert.ToInt32(txtPortNo.Text);

    // #1 ==> Create & Start The Listener
    tcpListener = new TcpListener(address, port);
    // Because I had socket reuse errors during step by step creation
    tcpListener.Server.SetSocketOption(SocketOptionLevel.Socket, SocketOptionName.ReuseAddress, true);

    //  #2 ==> Listener Begins Waiting For Connection
    tcpListener.Start();

    txtCommunicationTrace.Text += "Server>> " + "Waiting for Client Connection\r\n";
    btnStartServer.Hide();
    txtCommunicationTrace.Show();

    //  #3 ==> Create the Client Connection
    socketConnection = tcpListener.AcceptSocket();

    txtCommunicationTrace.Text += "Server>> " + "Client Connection Complete\r\n";
}
```

15] Getting better.



16] Should the user be able to type in the Communication Trace window? NO! Prevent them from doing so.

17] Lots of solutions. My choice is to send them to elsewhere any time they enter the Trace window.

18] Declare :

```
public partial class Server : Form
{
    Thread ReadClientMessage;
    TcpListener tcpListener;
    Socket socketConnection;
    bool ServerStartInitiated = false;
    bool ConnectionListening = false;
```

19] Change ConnectionListening here :

```
try
{
    //  IPAddress address = IPAddress.Parse("131.194.34.10");
    IPAddress address = IPAddress.Parse(lbIP.Text.ToString());
    int port = Convert.ToInt32(txtPortNo.Text);

    // #1 ==> Create & Start The Listener
    tcpListener = new TcpListener(address, port);
    // Because I had socket reuse errors during step by step creation
    tcpListener.Server.SetSocketOption(SocketOptionLevel.Socket, SocketOptionName.ReuseAddress, true);

    //  #2 ==> Listener Begins Waiting For Connection
    tcpListener.Start();
    ConnectionListening = true;

    txtCommunicationTrace.Text += "Server>> " + "Waiting for Client Connection\r\n";
    btnStartServer.Hide();
    txtCommunicationTrace.Show();

    //  #3 ==> Create the Client Connection
    socketConnection = tcpListener.AcceptSocket();

    txtCommunicationTrace.Text += "Server>> " + "Client Connection Complete\r\n";
}
```

20] Create an Enter event for the Communication Trace. Now we can not type in the Trace Window. Try it!

```
private void txtCommunicationTrace_Enter(object sender, EventArgs e)
{
    if (ConnectionListening)
        btnExit.Focus();
    else
        txtDataToSend.Focus();
}
```

21] Once you start listening, can you change the Port? YES Should you be able to? NO

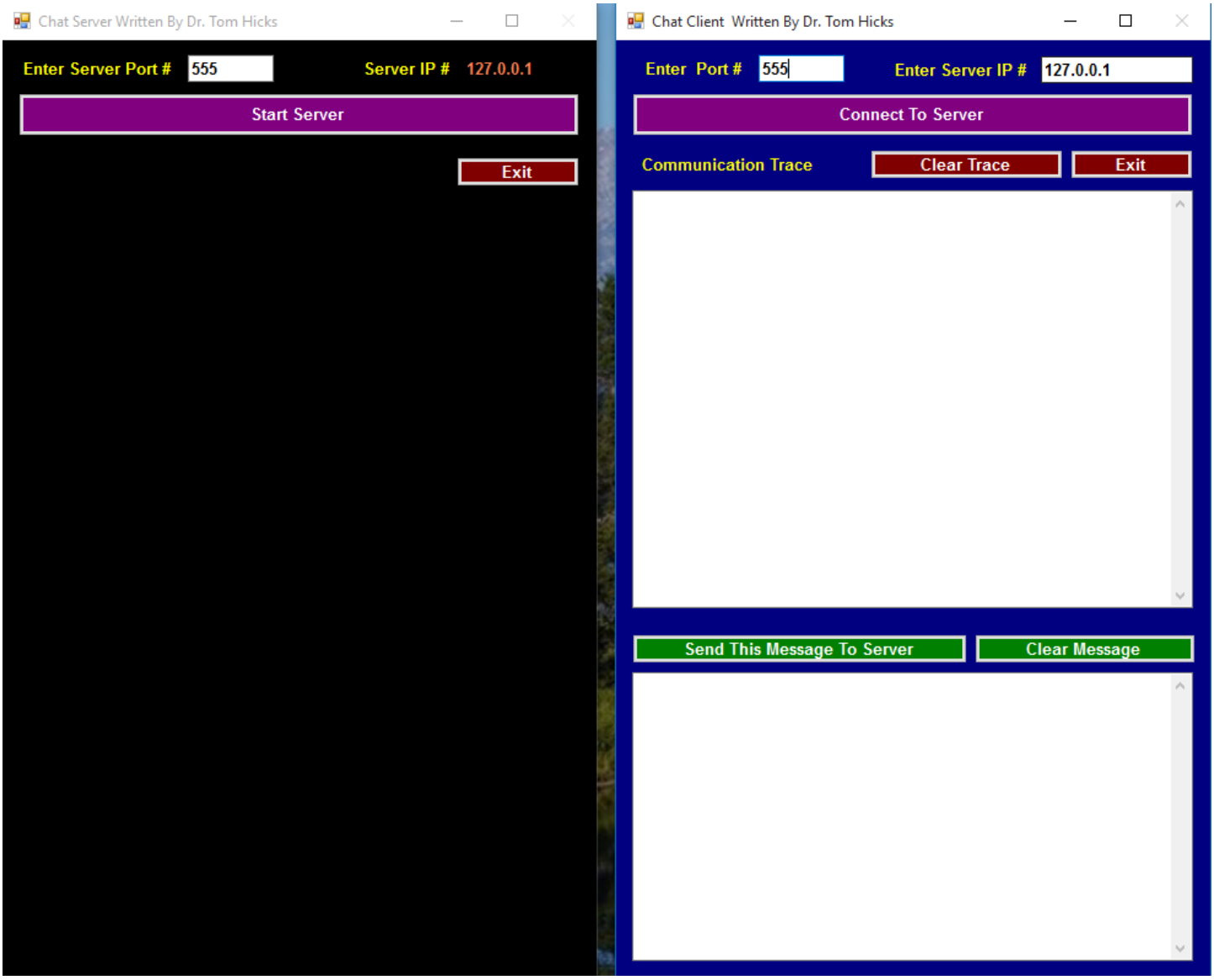
22] Create an Enter event for the Port No. Now we can not type in the Trace Window. Try it!

```
private void txtPortNo_Enter(object sender, EventArgs e)
{
    if (ConnectionListening)
        btnExit.Focus();
    else
        txtDataToSend.Focus();
}
```

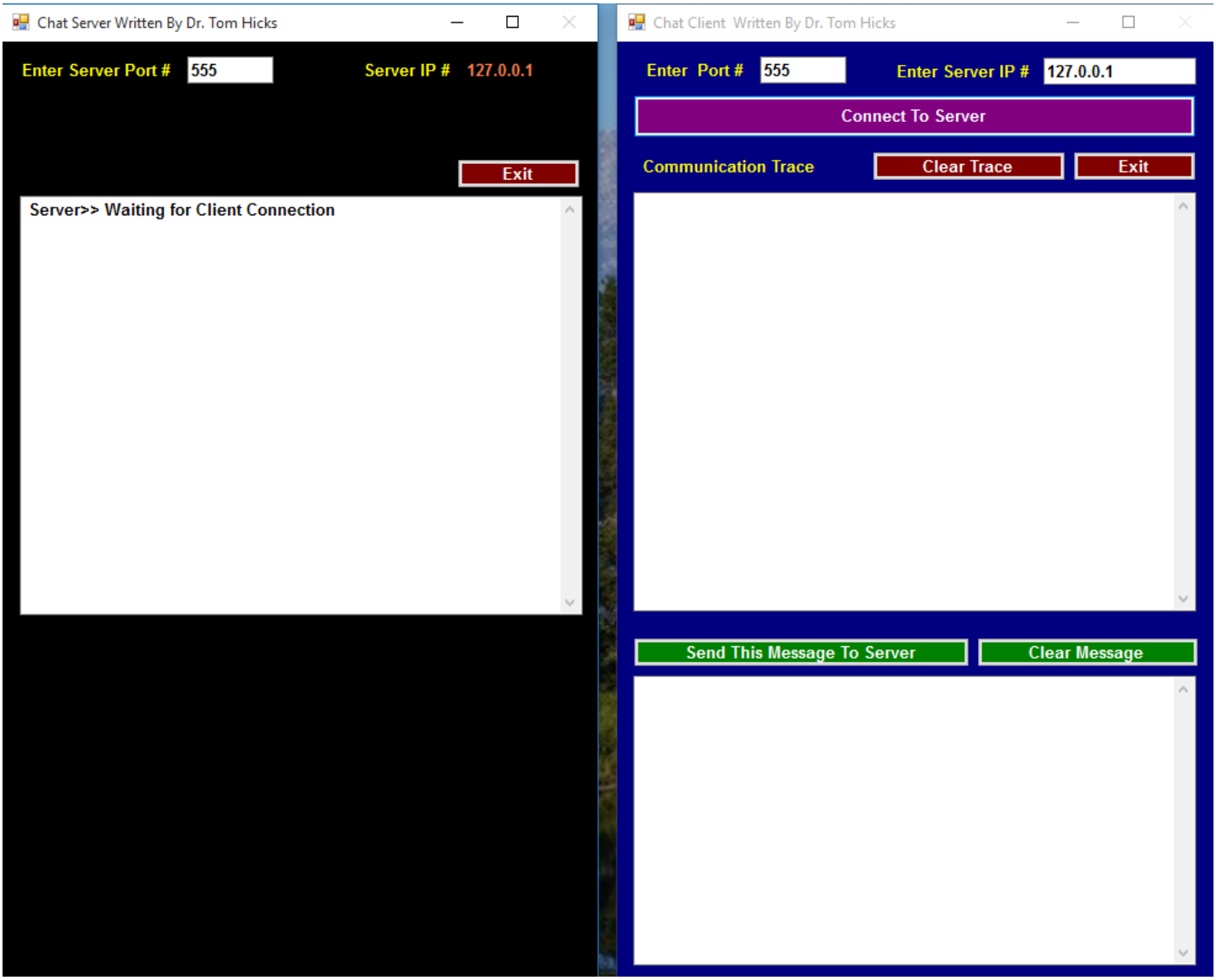
23] When do we show the panel?

```
try
{
    //  IPAddress address = IPAddress.Parse("131.194.34.10");
    IPAddress address = IPAddress.Parse(lbIP.Text.ToString());
    int port = Convert.ToInt32(txtPortNo.Text);

    // #1 ==> Create & Start The Listener
    tcpListener = new TcpListener(address, port);
    // Because I had socket reuse errors during step by step creation
    tcpListener.Server.SetSocketOption(SocketOptionLevel.Socket, SocketOptionName.ReuseAddress, true);

    //  #2 ==> Listener Begins Waiting For Connection
    tcpListener.Start();
    ConnectionListening = true;

    txtCommunicationTrace.Text += "Server>> " + "Waiting for Client Connection\r\n";
    btnStartServer.Hide();
    txtCommunicationTrace.Show();

    //  #3 ==> Create the Client Connection
    socketConnection = tcpListener.AcceptSocket();

    txtCommunicationTrace.Text += "Server>> " + "Client Connection Complete\r\n";
    panel1.Show();
}
```

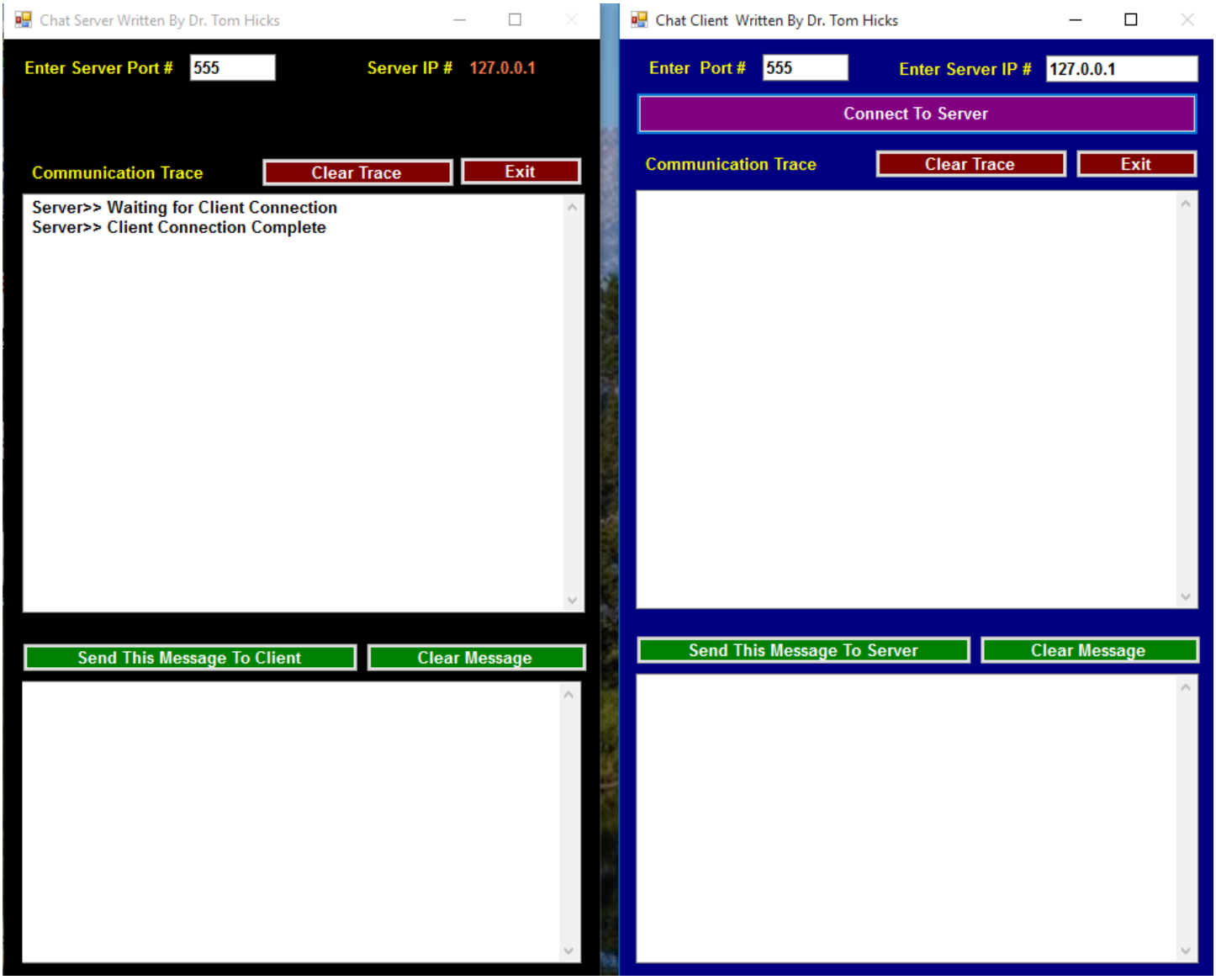
24] Program Started ➔ Port Set On Both ➔ We have work to do on the Client Interface



25] Server Listening ➔ We have work to do on the Client Interface



26] Connection Established ➔ We have work to do on the Client Interface



Do Part II On Client

Part III

#9 Prepare To Send/Receive Messages

1] Declare the following:

```
namespace Chat_Server
{
    public partial class Server : Form
    {
        Thread ReadClientMessage;
        TcpListener tcpListener;
        Socket socketConnection;
        bool ServerStartInitiated = false;
        bool ConnectionListening = false;

        NetworkStream serverNetworkStream;
        BinaryReader StreamReader;
        BinaryWriter StreamWriter;
```

2] Configure the Network Stream and the Stream Read/write mechanisms.

```
try
{
    //  IPAddress address = IPAddress.Parse("131.194.34.10");
    IPAddress address = IPAddress.Parse(lbIP.Text.ToString());
    int port = Convert.ToInt32(txtPortNo.Text);

    // #1 ==> Create & Start The Listener
    tcpListener = new TcpListener(address, port);
    // Because I had socket reuse errors during step by step creation
    tcpListener.Server.SetSocketOption(SocketOptionLevel.Socket, SocketOptionName.ReuseAddress, true);

    //  #2 ==> Listener Begins Waiting For Connection
    tcpListener.Start();
    ConnectionListening = true;

    txtCommunicationTrace.Text += "Server>> " + "Waiting for Client Connection\r\n";
    btnStartServer.Hide();
    txtCommunicationTrace.Show();

    //  #3 ==> Create the Client Connection
    socketConnection = tcpListener.AcceptSocket();

    // #4 ==> Create the Network Stream
    serverNetworkStream = new NetworkStream(socketConnection);
    // #5 ==> Create Stream Reader & Stream Writer for Network Stream
    StreamWriter = new BinaryWriter(serverNetworkStream);
    StreamReader = new BinaryReader(serverNetworkStream);

    txtCommunicationTrace.Text += "Server>> " + "Client Connection Complete\r\n";
    panel1.Show();
}
```

3] Compiles.

#10 Set Message

1] Add the following:

```
//==========================================================================
//==                Establish The Client Socket Connection                ==
//==========================================================================

if (ServerStartInitiated)
{
    ServerStartInitiated = false;
    try
    {
        //  IPAddress address = IPAddress.Parse("131.194.34.10");
        IPAddress address = IPAddress.Parse(lbIP.Text.ToString());
        int port = Convert.ToInt32(txtPortNo.Text);

        // #1 ==> Create & Start The Listener
        tcpListener = new TcpListener(address, port);
        // Because I had socket reuse errors during step by step creation
        tcpListener.Server.SetSocketOption(SocketOptionLevel.Socket, SocketOptionName.ReuseAddress, true);

        //  #2 ==> Listener Begins Waiting For Connection
        tcpListener.Start();
        ConnectionListening = true;

        txtCommunicationTrace.Text += "Server>> " + "Waiting for Client Connection\r\n";
        btnStartServer.Hide();
        txtCommunicationTrace.Show();

        //  #3 ==> Create the Client Connection
        socketConnection = tcpListener.AcceptSocket();

        // #4 ==> Create the Network Stream
        serverNetworkStream = new NetworkStream(socketConnection);

        // #5 ==> Create Stream Reader & Stream Writer for Network Stream
        StreamWriter = new BinaryWriter(serverNetworkStream);
        StreamReader = new BinaryReader(serverNetworkStream);

        txtCommunicationTrace.Text += "Server>> " + "Client Connection Complete\r\n";
        StreamWriter.Write("Server>> Your Connection Was Successful");
        panel1.Show();
        ConnectionListening = false;
    }

    catch (Exception error)
    {
        MessageBox.Show(error.Message.ToString());
    }
```

Do Part III On Client

Part IV

#11 Prepare To Receive Messages From Client

1] Declare the following:

```
public void RunChatServer()
{
    String ClientMessage = "";
    while (true)
    {
        //==========================================================================
        //==                Establish The Client Socket Connection                ==
        //==========================================================================
```

2] Write a loop to continue to read the Client Message & place it in the Trace

```
                    txtCommunicationTrace.Text += "Server>> " + "Client Connection Complete\r\n";
                    StreamWriter.Write("Server>> Your Connection Was Successful\r\n");
                    panel1.Show();
                    ConnectionListening = false;
                }

                catch (Exception error)
                {
                    MessageBox.Show(error.Message.ToString());
                }
            }

            //==========================================================================
            //==                      Read & Display From Client                      ==
            //==========================================================================
            do
            {
                try
                {
                    ClientMessage = StreamReader.ReadString();
                    txtCommunicationTrace.Text +=  ClientMessage + "\r\n";
                }

                // Handle Read Data Problem
                catch (Exception)
                {
                    break;
                }

            } while ((ClientMessage != "Client>> exit") && (socketConnection.Connected));
        }
    }
```

#12 Send Messages To Client & Update Trace

1] Add the following to button Send Message To Client

```
private void btnSendMessageToClient_Click(object sender, EventArgs e)
{
    StreamWriter.Write("Server>> " + txtDataToSend.Text.Trim());
    txtCommunicationTrace.Text += "\r\n" + "Server>> " + txtDataToSend.Text.Trim();
    txtDataToSend.Clear();
}
```

Do Part IV On Client

Final Touches

1] Write the code for Clear Trace

2] Write the code for Clear Message

3] Either use the Card-Grab lab as a reference, or search the Internet; create a function, called **GetIP** that explicitly returns a string with the IP address of the server.

4] Place a function call, in the page load, that fills lbIP with the Server IP.

5] If the server message is “exit”, send that message to the client and close the server.

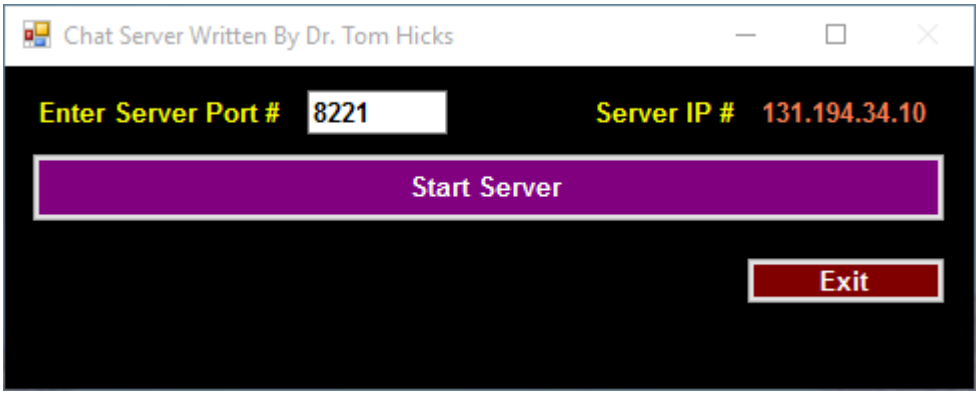
6] If the user pushes the exit button on the server, make sure the client does not crash.

7] Make sure that an invalid port entry does not crash the server.

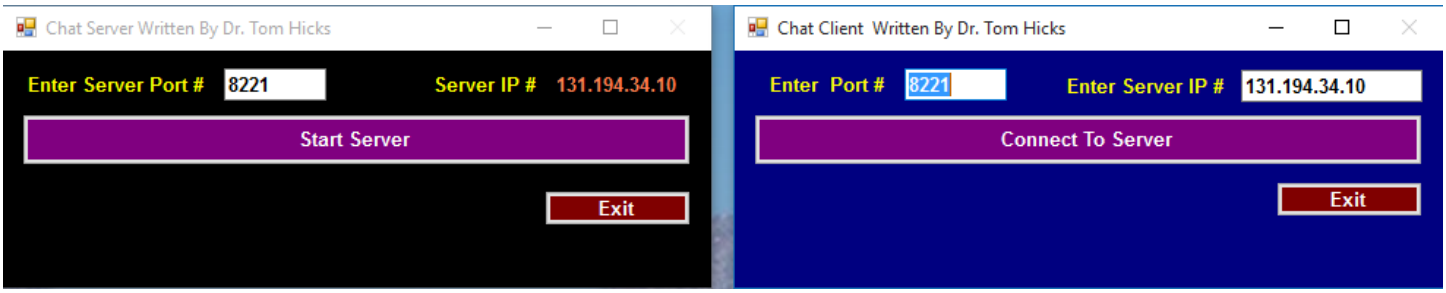
8] If the user enters a Port No that is not numeric, tell them that “The Port No Must Be Numeric” → keep them in the Port No until they set it correctly.

Trace Of My Solution

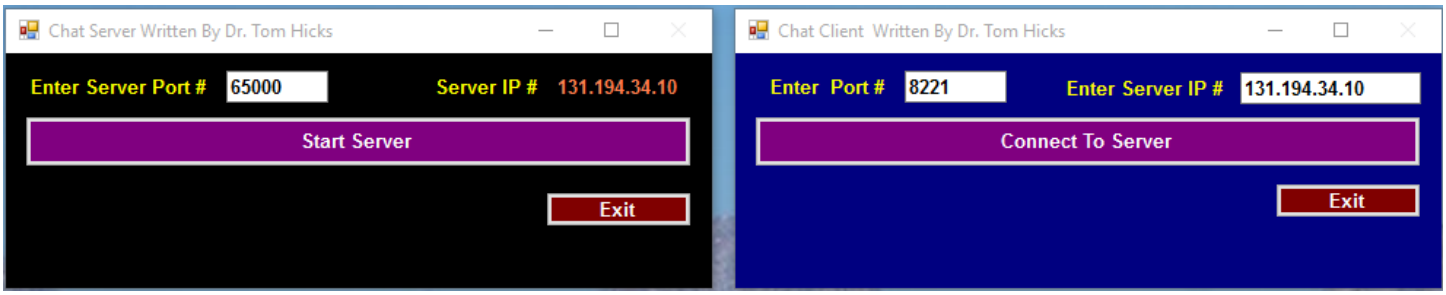
1] Start the Server



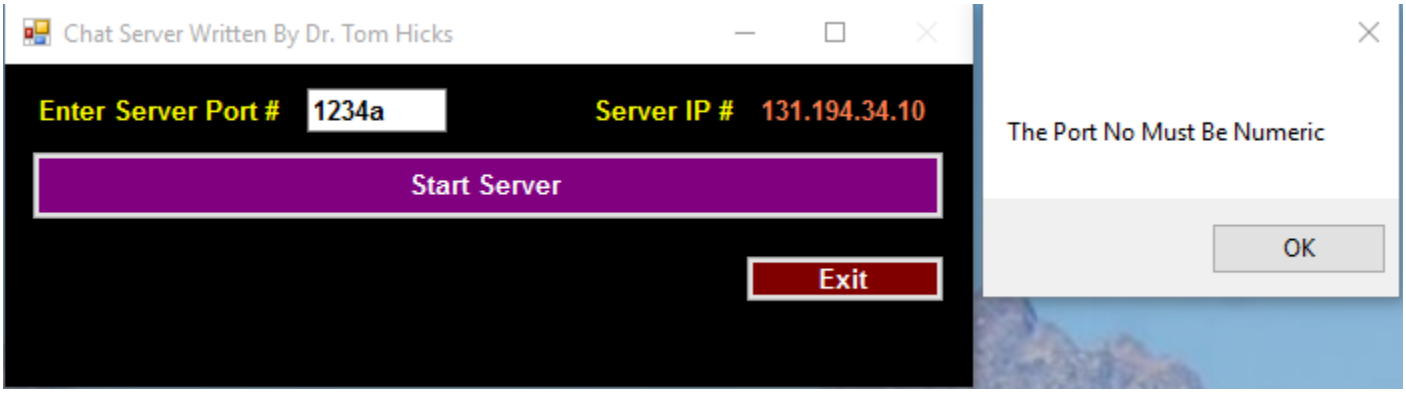
2] Start the Client



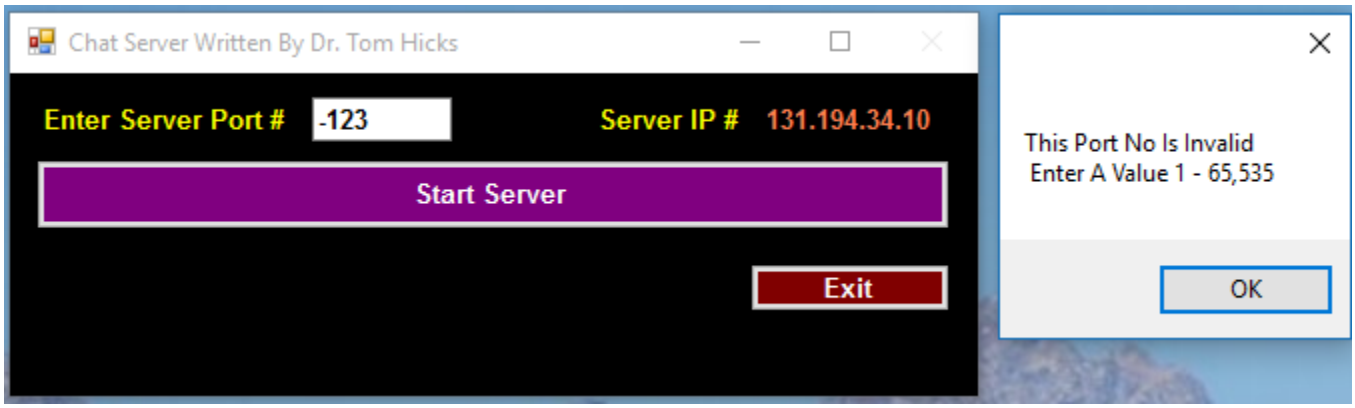
3] Server IP is automatic. **User changes the Port No to 65000**. The Server User has only three choices on form ➔ Change the Port No, Exit, and Start the Server.



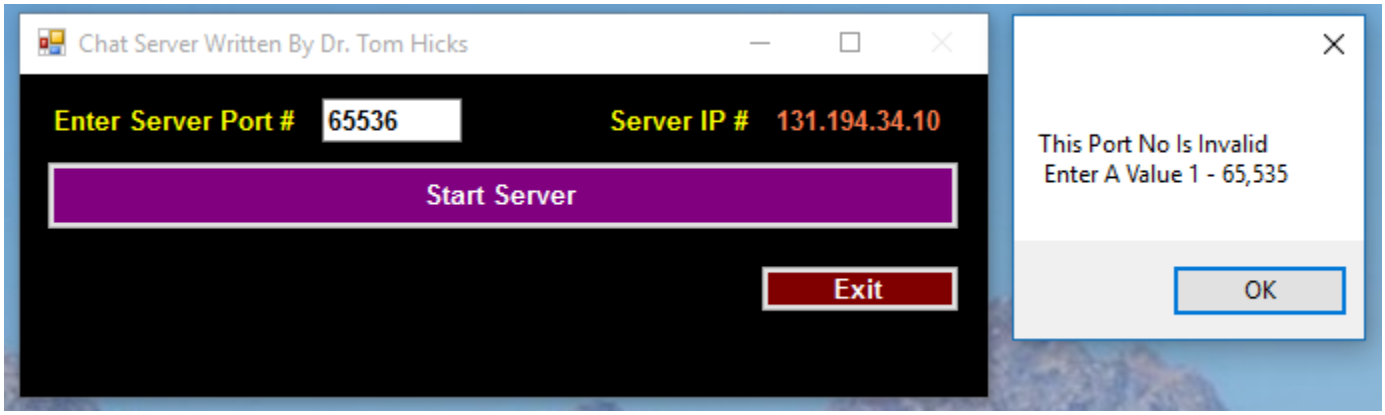
4] If the user enters a non-numeric Port No, then provide them the following error message. Note that they are returned to the Port No field to correct the problem.



5] If the user enters a Port No that is too low, then provide them the following error message. Note that they are returned to the Port No field to correct the problem.

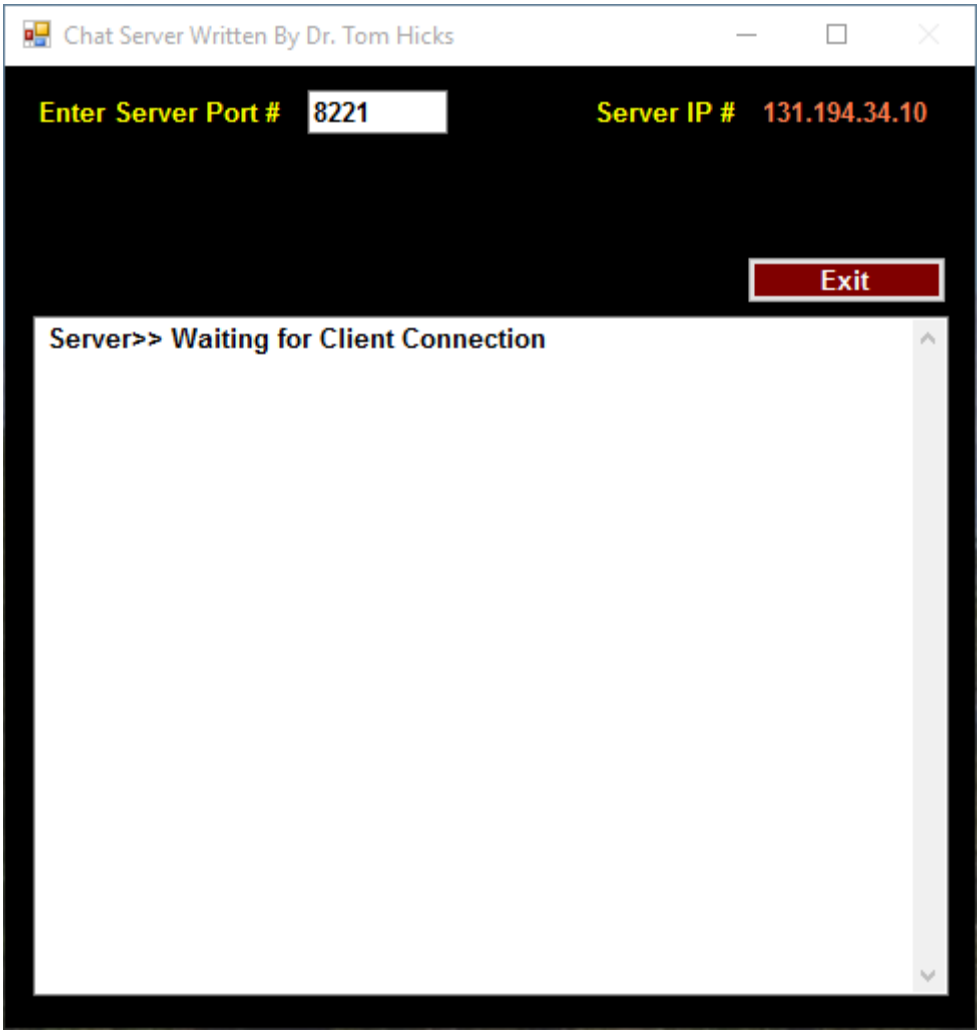


6] If the user enters a Port No that is too high, then provide them the following error message. Note that they are returned to the Port No field to correct the problem.



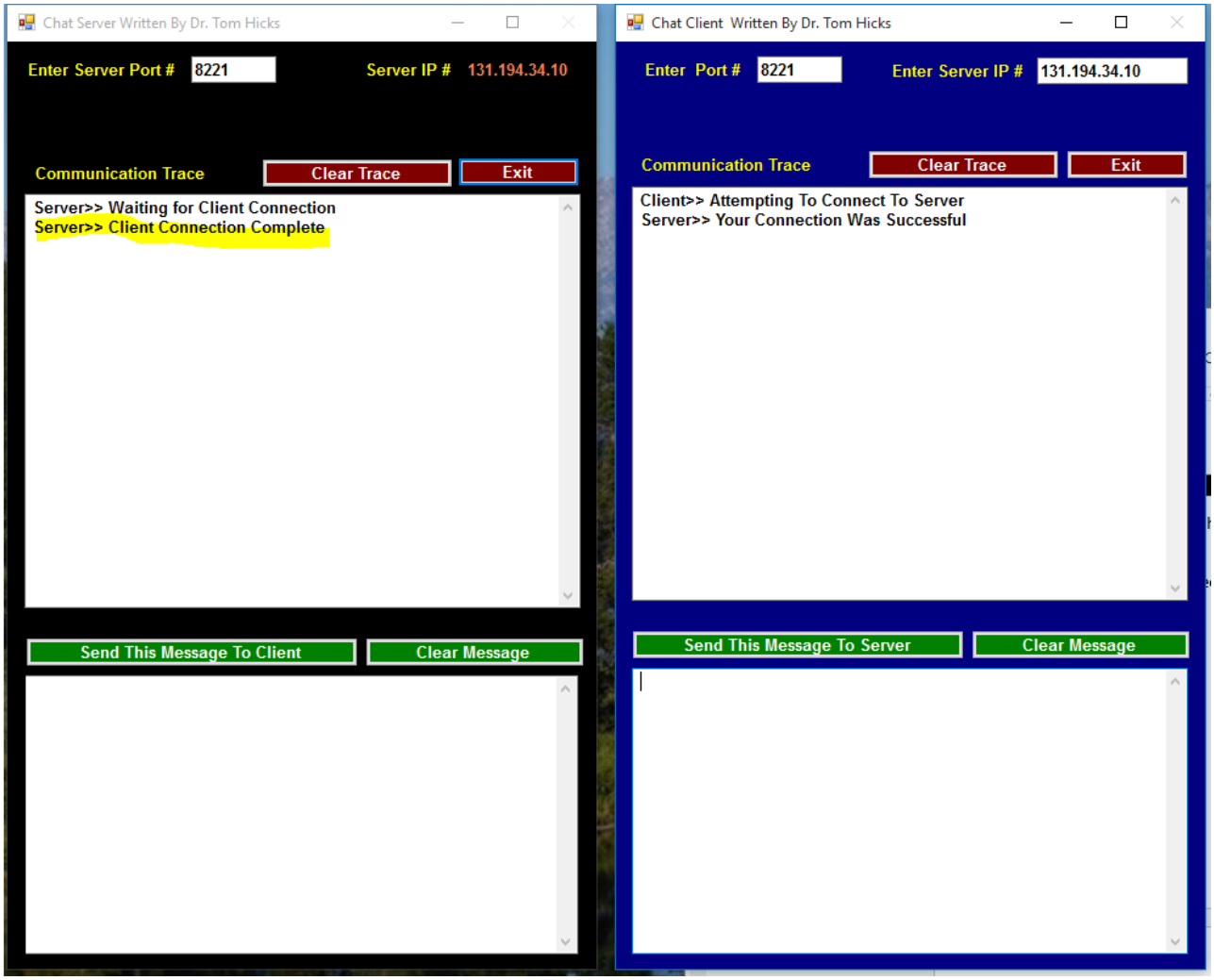
7] There should be no error created if the user chooses to Exit prior to starting the server.

8] Once the user starts the server, there is impossible to change the Port No.



9] Once the user starts the server, there is is a status message telling the user that the application is **"Waiting For The Client Server".**

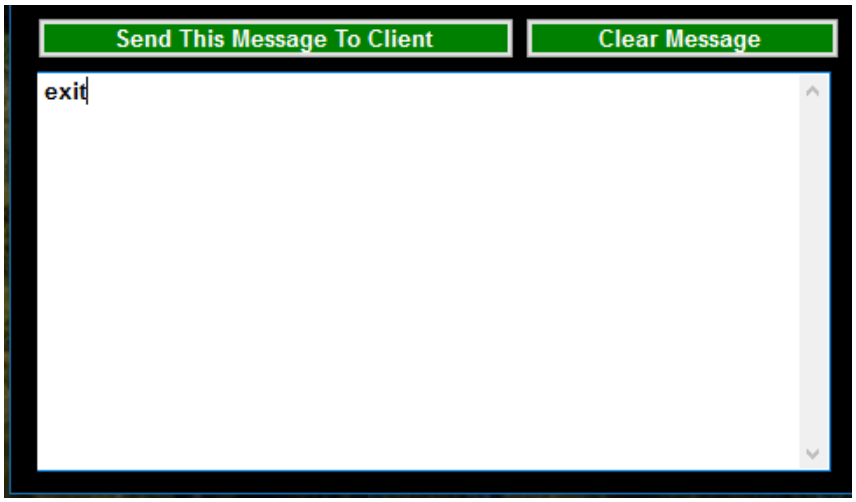
10] Once the client connects, the communication trace, on the server, will echo the message that the **"Client Connection Complete".**



11] Once the client is connected, the user can not change the Port No.

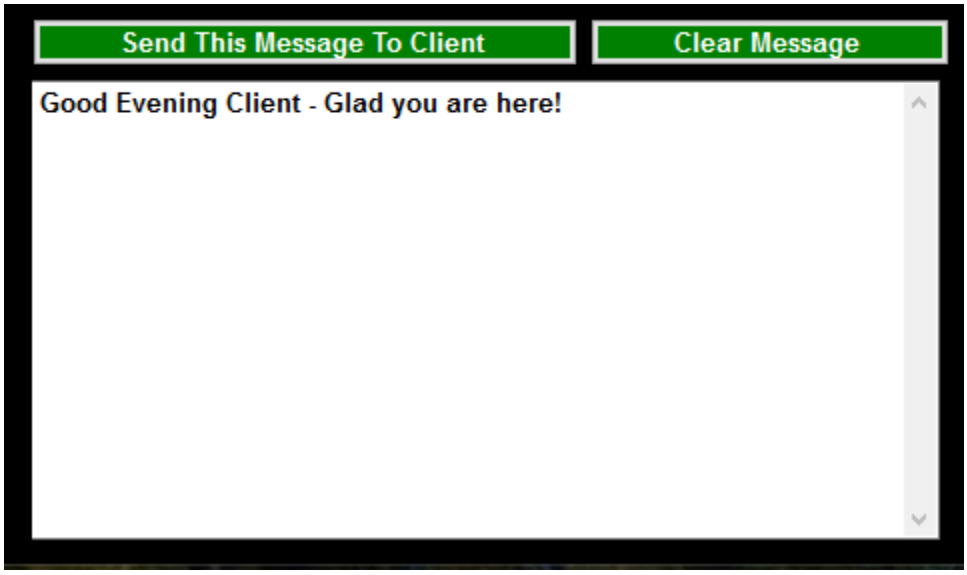
12] Once the client is connected, the user can not enter data into the txtCommunicationTrace.

13] Both the client and the server shall stop, with out error, when the user sends "exit" to the client.

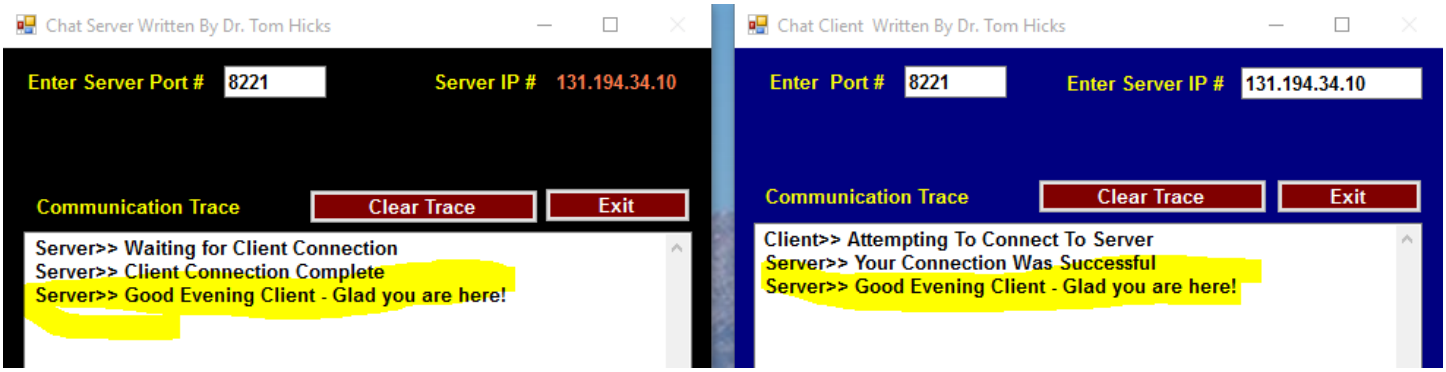


14] There should be no error created if the user chooses to Exit prior to starting the server. Both the client and the server shall stop, with out error.

15] When the server sends this message:



This message should be seen in Communication Trace of both the Server & the Client.



16] The Clear Message button erases all data written in **txtDataToSend**.

17] The Clear Trace button erases all data written in **txtCommunicationTrace**.