Socket Programming and Multithreading

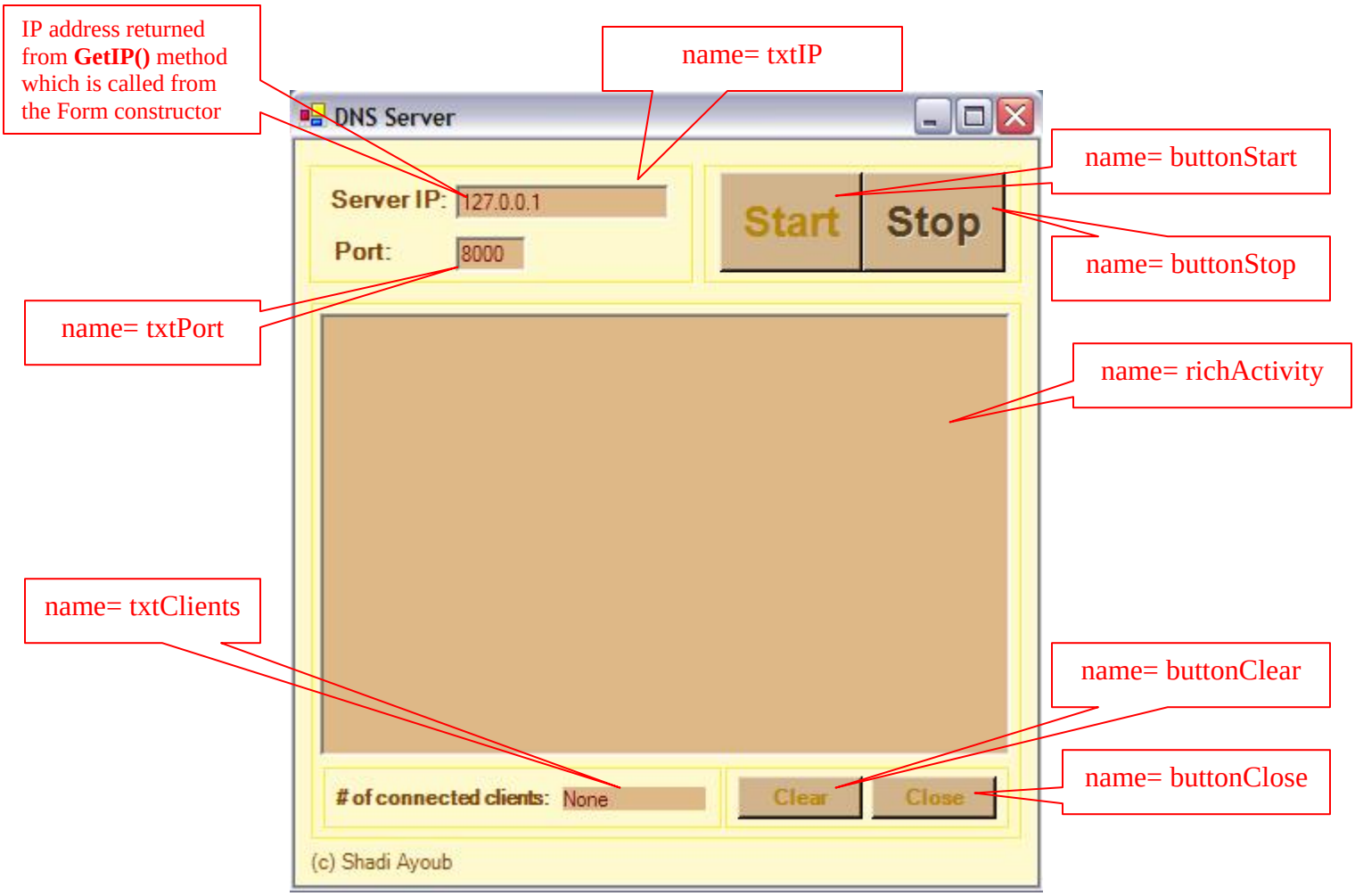
The DNS Server project

The purpose of this lesson is to show you how you can do socket programming in C#. Network programming in windows is possible with **sockets**. A socket is like a handle to a file. Socket programming resembles the file IO as does the Serial Communication. You can use sockets programming to have two applications communicate with each other. The two applications are typically on different computers but they can be on the same computer. For the two applications to talk to each other on the same or different computers using sockets, one application is generally a **server** that keeps listening to the incoming requests and the other application acts as a **client** and makes the connection to the server application. The server application can either accept or reject the connection. If the server accepts the connection, a dialog can begin with between the client and the server. Once the client is done with whatever it needs to do, it can close the connection with the server. Connections are expensive in the sense that servers allow finite connections to occur. During the time the client has an active connection; it can send the data to the server and/or receive the data.

The complexity begins here. When either side (client or server) sends data the other side is supposed to read the data. But how will the other side know when data has arrived. There are two options - either the application needs to poll for the data at regular intervals or there needs to be some sort of mechanism that would enable application to get notifications for the application to read the data at that time. Well, Windows is an event driven system and the notification system seems an obvious and best choice here.

So, the two applications that need to communicate with each other need to make a connection first. In order for the two applications to make connections, the two applications need to identify each other (or each other's computer). Computers on network have a unique identifier called IP address which is represented in dot-notation like 10.20.120.127, etc. Lets see how all this works in .NET.

1- The Server
    a. Implement the GUI shown below (call your project **DNServer**):



b. Include the following using statements:

- `using System.Net`
- `using System.Net.Sockets`
- `using System.Collections`
- `using System.Threading`
- `using System.IO`

c. declare the following variables, objects, and resources:

- `public delegate void UpdateRichEditCallback(string text);`
- `public AsyncCallback functionWorkerCallBack;`
- `private  Socket myMainSocket;`
- `private System.Collections.ArrayList myWorkerSocketList = ArrayList.Synchronized(new System.Collections.ArrayList());`
- `private int myClientCount = 0;`
- `private int myCurrentClientsCount = 0;`

d.

GetIP( )

```
String GetIP()
{
      String strHostName = Dns.GetHostName();

      // Find host by name
      IPHostEntry iphostentry = Dns.GetHostByName(strHostName);

      // Grab the first IP addresses
      String IPStr = "";
      foreach(IPAddress ipaddress in iphostentry.AddressList)
      {
            IPStr = ipaddress.ToString();
            return IPStr;
      }
      return IPStr;
}
```

e.

Constructor

```
public DNServer()
{
      // The InitializeComponent() call is required for Windows
      Forms designer support.
      InitializeComponent();

      // Display the local IP address on the GUI
      txtIP.Text = GetIP();
}
```

f. Test it!

g. Our socket communication will follow the following steps.
1) Create a socket
2) Bind the socket to an address or end point
3) Listen for an incoming communications attempt
4) Accept the communication
5) Send and receive messages (packets)
6) Shutdown the communication channel
7) Close the socket connection

buttonStart_Click( )

```
try
{
      // Check the port value
      if(txtPort.Text == "")
      {
            MessageBox.Show("Please specify a Port Number");
            return;
      }
      string portStr = txtPort.Text;
      int port = System.Convert.ToInt32(portStr);

      // Create the listening socket...
      myMainSocket = new Socket(AddressFamily.InterNetwork,
      SocketType.Stream, ProtocolType.Tcp);

      IPEndPoint ipLocal = new IPEndPoint (IPAddress.Any, port);

      // Bind to local IP Address...
      myMainSocket.Bind( ipLocal );
      // Start listening (waiting queue size = 4)...
      myMainSocket.Listen(4);
      // Create the call back for any client connections...
      myMainSocket.BeginAccept(new AsyncCallback (OnClientConnect),
      null);

      buttonStart.Enabled     = false;
      buttonStop.Enabled      = true;
}
catch(SocketException se)
{
      MessageBox.Show ( se.Message );
}
```

- SocketType.Stream: we would use reliable two way connection-based sockets. The other choice is to use unreliable connection-less sockets (SocketType.Dgram)
- IPEndPoint: A class under System.Net namespace which represents a network computer as an IP address and a port number. The IPEndPoint has two constructors - one that takes an IP Address and Port number and one that takes a long address and port number. Since we have computer IP address we would use the former.

h. Implement the call back function, which will be invoked when a client is connected

```
public void OnClientConnect(IAsyncResult asyn)
{
      try
      {
            // End the Asynchronous connection. This would
            // returns the reference to   a new Socket object
            Socket workerSocket = myMainSocket.EndAccept (asyn);

            // Now increment the client count for this client
            // in a thread safe manner
            Interlocked.Increment(ref myClientCount);
            Interlocked.Increment(ref myCurrentClientsCount);

            // Add the workerSocket reference to our ArrayList
            myWorkerSocketList.Add(workerSocket);

            //Show current number of clients on the GUI
            txtClients.Text = myCurrentClientsCount.ToString();

            // Send a welcome message to client
            string msg = "Welcome client" + myClientCount + "\n";
            SendMsgToClient(msg, myClientCount);

            string foo = "Client" + myClientCount + " Connected" + "\n";
            AppendToActivityRichBox(foo);

            // Let the worker Socket do the further processing for the
            // just connected client
            WaitForData(workerSocket, myClientCount);

            // Since the main Socket is now free, it can go back and wait for
            // other clients who are attempting to connect
            myMainSocket.BeginAccept(new AsyncCallback ( OnClientConnect ),null);
      }
      catch(ObjectDisposedException)
      {
            System.Diagnostics.Debugger.Log(0,"1","\n OnClientConnection: Socket
            has been closed\n");
      }
      catch(SocketException se)
      {
            MessageBox.Show ( se.Message );
      }
}
```

i. Implement the method that sends the message to the other side:

SendMsgToClient( )

```
void SendMsgToClient(string msg, int clientNumber)
{
      // Convert the reply to byte array
      byte[] byData = System.Text.Encoding.ASCII.GetBytes(msg);

      Socket workerSocket = (Socket)myWorkerSocketList[clientNumber - 1];
      workerSocket.Send(byData);
}
```

j. Implement the method that updates the RichTextBox which should be a thread safe method since it could be called by either the main thread or any of the worker threads:

```
private void AppendToActivityRichBox(string msg)
{
      // Check to see if this method is called from a thread
      // other than the one created the control
      if (InvokeRequired)
      {
            // We cannot update the GUI on this thread.
            // All GUI controls are to be updated by the main (GUI) thread.
            // Hence we will use the invoke method on the control which will
            // be called when the Main thread is free
            // Do UI update on UI thread
            object[] pList = {msg};
            richActivity.BeginInvoke(new UpdateRichEditCallback(OnUpdateRichEdit),
            pList);
      }
      else
      {
            // This is the main thread which created this control, hence update it
            // directly
            OnUpdateRichEdit(msg);
      }
}

private void OnUpdateRichEdit(string msg)
{
      richActivity.AppendText(msg);
      richActivity.Focus();
      richActivity.ScrollToCaret();
}
```

k. The method that Start waiting for data from the client

```
public void WaitForData(System.Net.Sockets.Socket soc, int clientNumber)
{
      try
      {
            If ( functionWorkerCallBack == null )
            {
                  // Specify the call back function which is to be
                  // invoked when there is any write activity by the
                  // connected client
                  functionWorkerCallBack = new AsyncCallback (OnDataReceived);
            }

            SocketPacket theSocPkt = new SocketPacket (soc, clientNumber);

            soc.BeginReceive (theSocPkt.dataBuffer, 0,
            theSocPkt.dataBuffer.Length,
            SocketFlags.None,
            functionWorkerCallBack,
            theSocPkt);
      }
      catch(SocketException se)
      {
            MessageBox.Show (se.Message );
      }
}

public class SocketPacket
{
      // Constructor which takes a Socket and a client number
      public SocketPacket(System.Net.Sockets.Socket socket, int clientNumber)
      {
            myCurrentSocket = socket;
            myClientNumber  = clientNumber;
      }
      public System.Net.Sockets.Socket myCurrentSocket;
      public int myClientNumber;
      // Buffer to store the data sent by the client
      public byte[] dataBuffer = new byte[1024];
}
```

```
public  void OnDataReceived(IAsyncResult asyn)
{
      SocketPacket socketData = (SocketPacket)asyn.AsyncState ;
      try
      {
            // Complete the BeginReceive() asynchronous call by EndReceive() method
            // which will return the number of characters written to the stream
            // by the client
            int iRx  = socketData.myCurrentSocket.EndReceive (asyn);
            char[] chars = new char[iRx +  1];
            // Extract the characters as a buffer
            System.Text.Decoder d = System.Text.Encoding.UTF8.GetDecoder();
            int charLen = d.GetChars(socketData.dataBuffer,
            0, iRx, chars, 0);

            System.String clientRequest = new System.String(chars);
            string msg = "Client" + socketData.myClientNumber + " Request: ";
            AppendToActivityRichBox(msg + clientRequest + "\n");

            //Open file to resolve domain
            string strFoo = "";
            string strResolved = "Unknown";
            string [] address;
            StreamReader srReadLine;
            try
            {
                  srReadLine = new StreamReader(
                  (System.IO.Stream)File.OpenRead("table.txt"),
                  System.Text.Encoding.ASCII);
            }
            catch( Exception exc )
            {
                  MessageBox.Show("Error: " + exc.Message );
                  return;
            }
            //Remove control characters (if any)!!!
            char [] myChar = clientRequest.ToCharArray();
            for( int j = 0; j < myChar.Length; j++ )
            {
                  if( Char.IsControl(myChar[j]) )
                  {
                        myChar[j] = ' ';
                  }
            }
            clientRequest = new System.String(myChar);
            clientRequest = clientRequest.Trim();

            //search for the host/IP address
            while( srReadLine.Peek() > -1 )
            {
                  strFoo = srReadLine.ReadLine();
                  address = strFoo.Split(',');
                  if( String.Compare(address[0], clientRequest) == 0 )
                  {
                        strResolved = address[1];
                        break;
                  }
                  else if( String.Compare(address[1], clientRequest) == 0 )
                  {
                        strResolved = address[0];
                        break;
                  }
            }
```

Continue..

```
            srReadLine.Close();
            msg = "Client" + socketData.myClientNumber + " Resolve: ";
            AppendToActivityRichBox(msg + strResolved + "\n");
            ///////
            // Send the result to the client
            // Convert the reply to byte array
            byte[] byData = System.Text.Encoding.ASCII.GetBytes(strResolved);

            Socket workerSocket = (Socket)socketData.myCurrentSocket;
            workerSocket.Send(byData);

            // Continue the waiting for data on the Socket
            WaitForData(socketData.myCurrentSocket, socketData.myClientNumber );
      }
      catch (ObjectDisposedException )
      {
            System.Diagnostics.Debugger.Log(0,"1","\nOnDataReceived: Socket has
            been closed\n");
      }
      catch(SocketException se)
      {
            string msg = "Client" + socketData.myClientNumber + " Disconnected" +
            "\n";
            AppendToActivityRichBox(msg);

            // Remove the reference to the worker socket of the closed client
            // so that this object will get garbage collected
            myWorkerSocketList[socketData.myClientNumber - 1] = null;

            //Thread-safe decrement
            Interlocked.Decrement(ref myCurrentClientsCount);

            //Show the count on the GUI
            txtClients.Text = myCurrentClientsCount.ToString();
            if( myCurrentClientsCount == 0 )
            {
                  txtClients.Text = "None";
            }
      }
}
```

l. Implement a function to close the server. The method should assure that all the opened sockets are closed:

```
void CloseSockets()
{
      if(myMainSocket != null)
      {
            myMainSocket.Close();
      }
      Socket workerSocket = null;
      for(int i = 0; i < myWorkerSocketList.Count; i++)
      {
            workerSocket = (Socket)myWorkerSocketList[i];
            if(workerSocket != null)
            {
                  workerSocket.Close();
                  workerSocket = null;
            }
      }
}
```

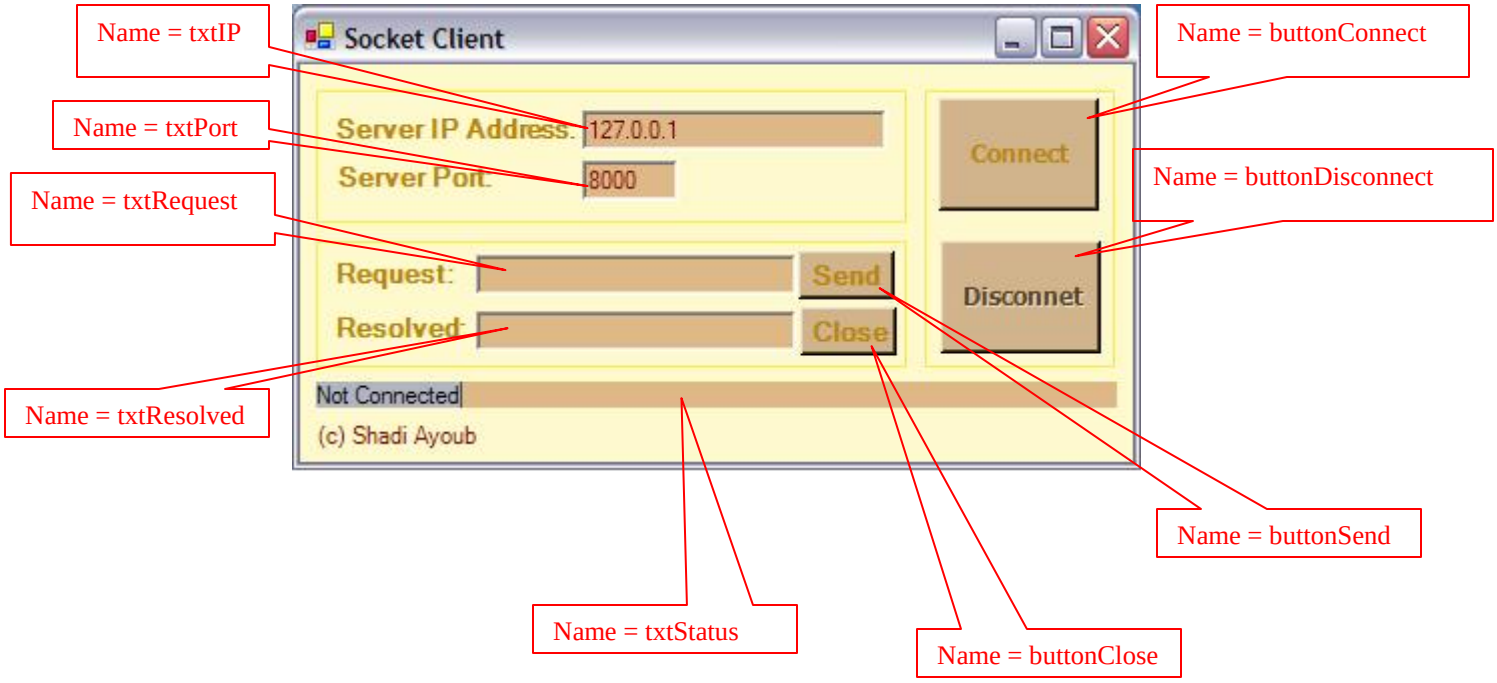
m. Implement the rest of methods:

```
private void buttonClear_Click(object sender, System.EventArgs e)
{
      richActivity.Clear();
}

void ButtonCloseClick(object sender, System.EventArgs e)
{
      CloseSockets();
      Close();
}

void ButtonStopClick(object sender, System.EventArgs e)
{
      CloseSockets();
      buttonStart.Enabled = true;
      buttonStop.Enabled      = false;
      txtClients.Text = "None";
}
```

n. Save the `table.txt` file in the Debug directory

2- The client

a. Design the GUI for the client side (call your project `SocketClient`)



b. Include the following using statements:

- `using System.Net`
- `using System.Net.Sockets`

c. declare the following variables, objects, and resources:

- `byte[] myDataBuffer = new byte [10];`
- `IAsyncResult myResult;`
- `public AsyncCallback functionCallBack ;`
- `public Socket myClientSocket;`

d.

GetIP( )

```
String GetIP()
{
      String strHostName = Dns.GetHostName();

      // Find host by name
      IPHostEntry iphostentry = Dns.GetHostByName(strHostName);

      // Grab the first IP addresses
      String IPStr = "";
      foreach(IPAddress ipaddress in iphostentry.AddressList)
      {
            IPStr = ipaddress.ToString();
            return IPStr;
      }
      return IPStr;
}
```

e.

Constructor

```
public SocketClient()
{
	// The InitializeComponent() call is required for Windows
	Forms designer support.
	InitializeComponent();
	txtIP.Text = GetIP();
}
```

f. Test it!

g.

```
void ButtonConnectClick(object sender, System.EventArgs e)
{
	// See if we have text on the IP and Port text fields
	if(txtIP.Text == "" || txtPort.Text == "")
	{
		MessageBox.Show("IP Address and Port Number are required to
		connect to the Server\n");
		return;
	}
	try
	{
		UpdateControls(false);
		// Create the socket instance
		myClientSocket = new Socket (AddressFamily.InterNetwork,
		SocketType.Stream, ProtocolType.Tcp );

		// Cet the remote IP address
		IPAddress ip = IPAddress.Parse (txtIP.Text);
		int iPortNo = System.Convert.ToInt16 ( txtPort.Text);
		// Create the end point
		IPEndPoint ipEnd = new IPEndPoint (ip,iPortNo);
		// Connect to the remote host
		myClientSocket.Connect ( ipEnd );
		if(myClientSocket.Connected)
		{
			UpdateControls(true);
			//Wait for data asynchronously
			WaitForData();
		}
	}
	catch(SocketException se)
	{
		string str;
		str = "\nConnection failed: " + se.Message;
		MessageBox.Show (str);
		UpdateControls(false);
	}
}
```

h.

```
public void WaitForData()
{
      try
      {
            if  ( functionCallBack == null )
            {
                  functionCallBack = new AsyncCallback (OnDataReceived);
            }
            SocketPacket theSocPkt = new SocketPacket();
            theSocPkt.thisSocket = myClientSocket;
            // Start listening to the data asynchronously
            myResult = myClientSocket.BeginReceive (theSocPkt.dataBuffer,
            0, theSocPkt.dataBuffer.Length,
            SocketFlags.None,
            functionCallBack,
            theSocPkt);
      }
      catch(SocketException se)
      {
            MessageBox.Show (se.Message );
      }
}
```

```
public class SocketPacket
{
      public System.Net.Sockets.Socket
      thisSocket;
      public byte[] dataBuffer = new
      byte[1024];
}
```

```
private void UpdateControls( bool connected )
{
      buttonConnect.Enabled = !connected;
      buttonDisconnect.Enabled = connected;
      string connectStatus = connected? "Connected" :
      "Not Connected";
      txtStatus.Text = connectStatus;
}
```

```
public  void OnDataReceived(IAsyncResult asyn)
{
      try
      {
            SocketPacket theSockId = (SocketPacket)asyn.AsyncState ;
            int iRx  = theSockId.thisSocket.EndReceive (asyn);
            char[] chars = new char[iRx +  1];
            System.Text.Decoder d = System.Text.Encoding.UTF8.GetDecoder();
            int charLen = d.GetChars(theSockId.dataBuffer, 0, iRx, chars, 0);
            System.String szData = new System.String(chars);
            txtResolved.Text = szData;
            WaitForData();
      }
      catch (ObjectDisposedException )
      {
            System.Diagnostics.Debugger.Log(0,"1","\nOnDataReceived: Socket has been
            closed\n");
      }
      catch(SocketException se)
      {
            MessageBox.Show (se.Message );
      }
}
```

```
void ButtonDisconnectClick(object sender, System.EventArgs e)
{
      if ( myClientSocket != null )
      {
            myClientSocket.Close();
            myClientSocket = null;
            UpdateControls(false);
      }
}
private void SocketClient_Load(object sender, System.EventArgs e)
{
      txtRequest.Focus();
}

void ButtonCloseClick(object sender, System.EventArgs e)
{
      if ( myClientSocket != null )
      {
            myClientSocket.Close ();
            myClientSocket = null;
      }
      Close();
}
```

```
void ButtonSendMessageClick(object sender, System.EventArgs e)
{
	try
	{
		string msg = txtRequest.Text;
		if( msg.Trim() == "" )
		{
			MessageBox.Show ("Empty textbox");
			return;
		}
		// New code to send strings
		NetworkStream networkStream = new NetworkStream(myClientSocket);
		System.IO.StreamWriter streamWriter = new
		System.IO.StreamWriter(networkStream);
		streamWriter.WriteLine(msg);
		streamWriter.Flush();

	}
	catch(SocketException se)
	{
		MessageBox.Show (se.Message );
	}
}
```