Asynchronous Socket (Client / Server) example.

Example

Server Side example

Create Listener for server

Start of with creating an server that will handle clients that connect, and requests that will be send. So create an Listener Class that will handle this.

```
class Listener
{
    public Socket ListenerSocket; //This is the socket that will listen to any incoming connections
    public short Port = 1234; // on this port we will listen

    public Listener()
    {
        ListenerSocket = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);
    }
 }
```

First we need to initialize the Listener socket where we can listen on for any connections. We are going to use an Tcp Socket that is why we use SocketType.Stream. Also we specify to witch port the server should listen to

Then we start listening for any incoming connections.

The tree methods we use here are:

1. ListenerSocket.Bind();

   This method binds the socket to an IPEndPoint. This class contains the host and local or remote port information needed by an application to connect to a service on a host.

2. ListenerSocket.Listen(10);

   The backlog parameter specifies the number of incoming connections that can be queued for acceptance.

3. ListenerSocket.BeginAccept();

   The server will start listening for incoming connections and will go on with other logic. When there is an connection the server switches back to this method and will run the AcceptCallBack methodt

```
public void StartListening()
{
    try
    {
            MessageBox.Show($"Listening started port:{Port} protocol type: {ProtocolType.Tcp}");
            ListenerSocket.Bind(new IPEndPoint(IPAddress.Any, Port));
            ListenerSocket.Listen(10);
            ListenerSocket.BeginAccept(AcceptCallback, ListenerSocket);
    }
    catch(Exception ex)
    {
        throw new Exception("listening error" + ex);
    }
}
```

So when a client connects we can accept them by this method:

Three methods wee use here are:

1. ListenerSocket.EndAccept()

   We started the callback with `Listener.BeginAccept()` end now we have to end that call back. `The EndAccept()` method accepts an IAsyncResult parameter, this will store the state of the asynchronous method, From this state we can extract the socket where the incoming connection was coming from.

2. `ClientController.AddClient()`

   With the socket we got from `EndAccept()` we create an Client with an own made method *(code ClientController below server example)*.

3. ListenerSocket.BeginAccept()

   We need to start listening again when the socket is done with handling the new connection. Pass in the method who will catch this callback. And also pass int the Listener socket so we can reuse this socket for upcoming connections.

```
public void AcceptCallback(IAsyncResult ar)
{
    try
    {
        Console.WriteLine($"Accept CallBack port:{Port} protocol type: {ProtocolType.Tcp}");
        Socket acceptedSocket = ListenerSocket.EndAccept(ar);
        ClientController.AddClient(acceptedSocket);

        ListenerSocket.BeginAccept(AcceptCallback, ListenerSocket);
    }
    catch (Exception ex)
    {
        throw new Exception("Base Accept error"+ ex);
    }
}
```

Now we have an Listening Socket but how do we receive data send by the client that is what the next code is showing.

Create Server Receiver for each client

First of create a receive class with a constructor that takes in a Socket as parameter:

```
public class ReceivePacket
{
    private byte[] _buffer;
    private Socket _receiveSocket;

    public ReceivePacket(Socket receiveSocket)
    {
       _receiveSocket = receiveSocket;
    }
}
```

In the next method we first start off with giving the buffer a size of 4 bytes (Int32) or package contains to parts {lenght, actual data}. So the first 4 bytes we reserve for the lenght of the data the rest for the actual data.

Next we use BeginReceive() method. This method is used to start receiving from connected clients and when it will receive data it will run the `ReceiveCallback` function.

```
    public void StartReceiving()
    {
        try
        {
            _buffer = new byte[4];
            _receiveSocket.BeginReceive(_buffer, 0, _buffer.Length, SocketFlags.None, ReceiveCallback, null);
        }
        catch {}
    }

    private void ReceiveCallback(IAsyncResult AR)
    {
        try
        {
            // if bytes are less than 1 takes place when a client disconnect from the server.
            // So we run the Disconnect function on the current client
            if (_receiveSocket.EndReceive(AR) > 1)
            {
                // Convert the first 4 bytes (int 32) that we received and convert it to an Int32 (this is the
size for the coming data).
                _buffer = new byte[BitConverter.ToInt32(_buffer, 0)];
                // Next receive this data into the buffer with size that we did receive before
                _receiveSocket.Receive(_buffer, _buffer.Length, SocketFlags.None);
                // When we received everything its onto you to convert it into the data that you've send.
                // For example string, int etc... in this example I only use the implementation for sending and
receiving a string.

                // Convert the bytes to string and output it in a message box
                string data = Encoding.Default.GetString(_buffer);
                MessageBox.Show(data);
                // Now we have to start all over again with waiting for a data to come from the socket.
                StartReceiving();
            }
            else
            {
                Disconnect();
            }
        }
        catch
        {
            // if exeption is throw check if socket is connected because than you can startreive again else
Dissconect
            if (!_receiveSocket.Connected)
            {
                Disconnect();
            }
            else
            {
                StartReceiving();
            }
        }
    }

    private void Disconnect()
    {
        // Close connection
        _receiveSocket.Disconnect(true);
        // Next line only apply for the server side receive
        ClientController.RemoveClient(_clientId);
        // Next line only apply on the Client Side receive
        Here you want to run the method TryToConnect()
    }
```

So we've setup a server that can receive and listen for incoming connections. When a clients connect it will be added to a list of clients and every client has his own receive class. To make the server listen:

```
Listener listener = new Listener();
listener.StartListening();
```

Some Classes I use in this example

```
    class Client
    {
        public Socket _socket { get; set; }
        public ReceivePacket Receive { get; set; }
        public int Id { get; set; }

        public Client(Socket socket, int id)
        {
            Receive = new ReceivePacket(socket, id);
            Receive.StartReceiving();
            _socket = socket;
            Id = id;
        }
    }

     static class ClientController
     {
          public static List<Client> Clients = new List<Client>();

          public static void AddClient(Socket socket)
          {
              Clients.Add(new Client(socket,Clients.Count));
          }

          public static void RemoveClient(int id)
          {
              Clients.RemoveAt(Clients.FindIndex(x => x.Id == id));
          }
      }
```

Client Side example

Connecting to server

First of all we want to create a class what connects to the server te name we give it is: Connector:

```
class Connector
{
    private Socket _connectingSocket;
}
```

Next Method for this class is TryToConnect()

This method goth a few interestin things:

1. Create the socket;
2. Next I loop until the socket is connected
3. Every loop it is just holding the Thread for 1 second we don't want to DOS the server XD
4. With Connect() it will try to connect to the server. If it fails it will throw an exception but the wile will keep the program connecting to the server. You can use a Connect CallBack method for this, but I'll just go for calling a method when the Socket is connected.
5. Notice the Client is now trying to connect to your local pc on port 1234.

```
public void TryToConnect()
{
    _connectingSocket = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);

     while (!_connectingSocket.Connected)
     {
         Thread.Sleep(1000);

         try
         {
             _connectingSocket.Connect(new IPEndPoint(IPAddress.Parse("127.0.0.1"), 1234));
         }
         catch { }
     }
     SetupForReceiveing();
    }
}

private void SetupForReceiveing()
{
   // View Client Class bottom of Client Example
    Client.SetClient(_connectingSocket);
    Client.StartReceiving();
}
```

Sending a message to the server

So now we have an almost finish or Socket application. The only thing that we don't have jet is a Class for sending a message to the server.

```
public class SendPacket
{
    private Socket _sendSocked;

    public SendPacket(Socket sendSocket)
    {
        _sendSocked = sendSocket;
    }

    public void Send(string data)
    {
        try
        {
            /* what hapends here:
                 1. Create a list of bytes
                 2. Add the length of the string to the list.
                    So if this message arrives at the server we can easily read the length of the coming message.
                 3. Add the message(string) bytes
            */

            var fullPacket = new List<byte>();
            fullPacket.AddRange(BitConverter.GetBytes(data.Length));
            fullPacket.AddRange(Encoding.Default.GetBytes(data));

            /* Send the message to the server we are currently connected to.
            Or package stucture is {length of data 4 bytes (int32), actual data}*/
            _sendSocked.Send(fullPacket.ToArray());
        }
        catch (Exception ex)
        {
            throw new Exception();
        }
    }
```

Finaly crate two buttons one for connect and the other for sending a message:

```
private void ConnectClick(object sender, EventArgs e)
{
    Connector tpp = new Connector();
    tpp.TryToConnect();
}

private void SendClick(object sender, EventArgs e)
{
    Client.SendString("Test data from client");
}
```

The client class I used in this example

```
public static void SetClient(Socket socket)
{
    Id = 1;
    Socket = socket;
    Receive = new ReceivePacket(socket, Id);
    SendPacket = new SendPacket(socket);
}
```

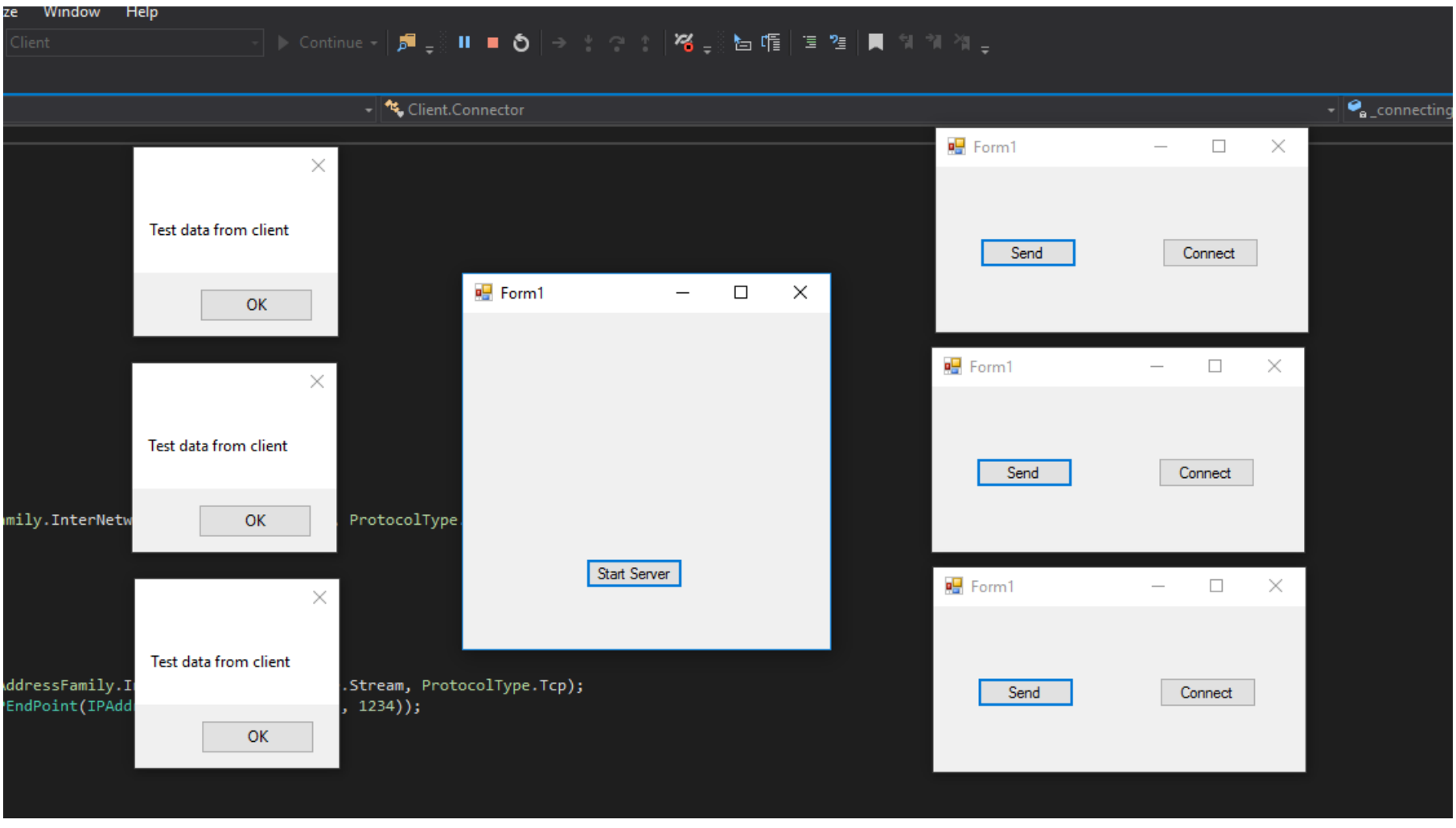
Notice

The Receive Class from the server is the same as the receive class from the client.

Conclusion

You now have a server and a client. You can work this basic example out. For example make it that the server also can receive files or other tings. Or send a message to the client. In the server you got a list of client so when you receive something you will know from with client it came from.

Final result:



Previous Next