



Articles » General Programming » Internet / Network » Client/Server Development

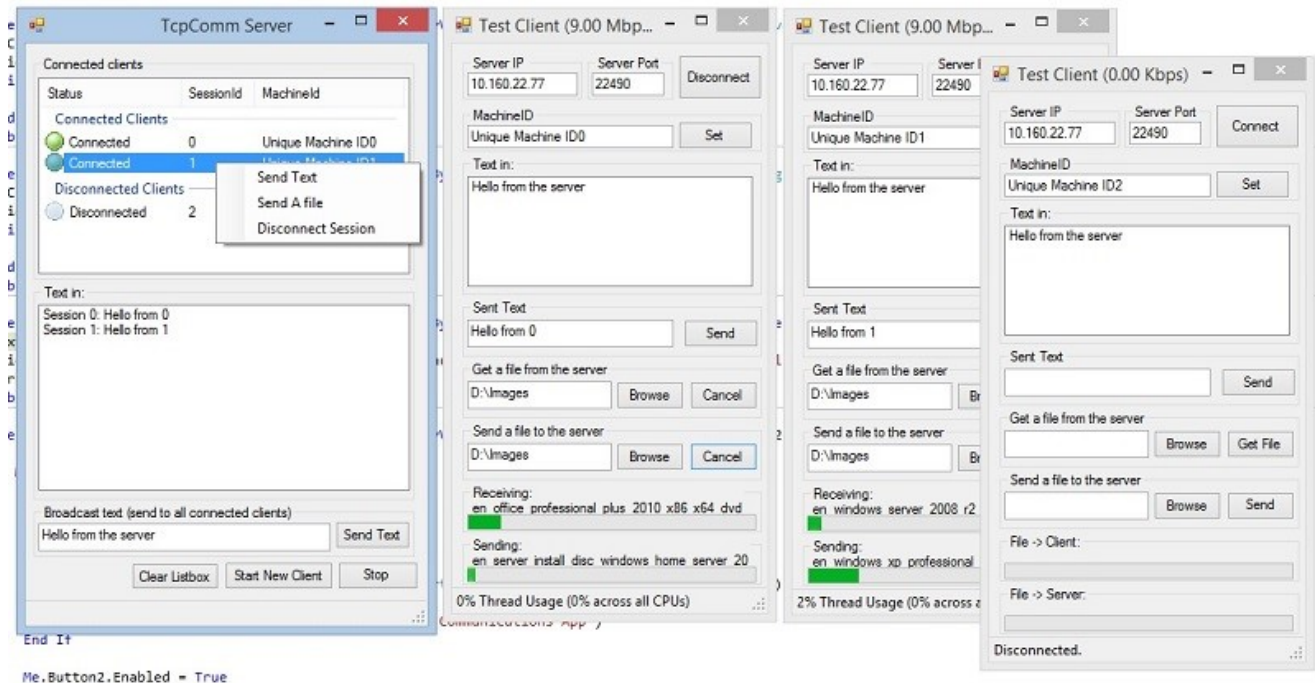
Reusable Multithreaded TCP/IP Client and Server Classes with Example Project in VB.NET

 **pdxtader**
13 Nov 2019 CPOL

A multithreaded server class that accepts multiple connections from a provided client class. Each client can send and receive files and text (byte data) simultaneously along 250 available channels.

[Download TcpCommExampleProjectV4.08 - 342.3 KB](#)

[Download TcpCommExampleProjectV2 - 41.1 KB](#)



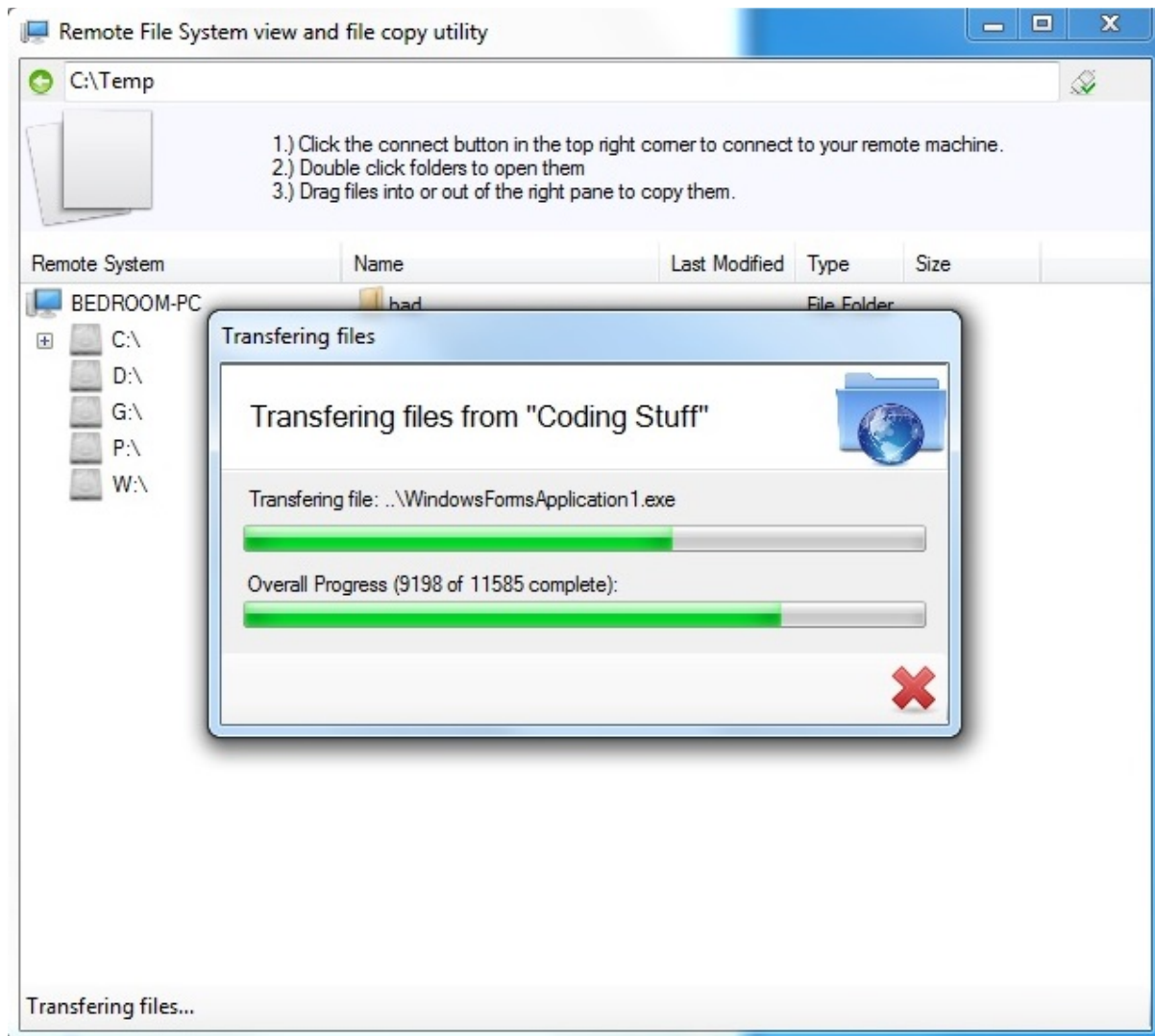
Introduction

Please see below for a list of fixes / changes in version 4.0. I've left version 2 up so it's available for people who may still need a .NET 2.0 only solution. This latest version requires .NET 4 or higher. Version 4.0 is not compatible with previous versions!

This example project contains two classes - **TcpCommServer** and **TcpCommClient**. With these classes, you will not only be able to instantly add TCP/IP functionality to your VB.NET applications, but it also has most of the bells and whistles we're all looking for. With these classes, you will be able to connect multiple clients to the server on the same port. You will be able to easily: throttle bandwidth to the clients, and send and receive files and data (text?) along 250 provided channels simultaneously on a single connection.

Looking for a Way to Transfer Folders Fast?

I put together an example application on how to use this library to transfer folders. The example application actually provides fast FTP like functionality, and comes with an explorer like window. Transfers are begun via drag and drop:



Background

When I first started looking for information or example code for TCP/IP communication in VB.NET, I have to admit I was looking for it all. I was willing to accept the bits I needed to get started, but I was hoping, like you probably are, that I would find it all... some easily transportable classes that would allow me to do what I think most of us want when we first start looking into this - send and receive files and our own data or text, all over a single connection, simultaneously. It's what we'll all need eventually, for one project or another, when the time comes to build a client/server application. We'll also need to control the bandwidth at the server, or someone will use it all up downloading a large file... And oh - it would be nice if we could connect to the server on the same port from multiple clients, right?

I looked for a long time and never found anything right out of the box that would do what I needed, so I decided to build it myself. Along the way, I began to understand why people don't just freely give this kind of work away. It took me quite a bit of time coding, troubleshooting, and testing this - but it is, after all, just a tool. It's what we do with it that will be worth something in the end.

So here it is.

Channels

We want to be able to do everything... right? And all over a single connection. You want to be able to send, say, a stream of screenshots, or video, or text, and not have to code your own way of separating one from another. The channel was my answer. Every time you use the **SendBytes** method, you send that data with a channel identifier. When it comes out the other end in the callback, it arrives with the channel identifier you sent it with so you can tell your screenshot bytes from your text - if that's what you're sending. As I said above, you have 250 channels at your disposal (1 to 251).

OK, on to some code.

Using the Code

First of all, both classes use delegates to do callbacks. All you have to do is pass the **AddressOf** YourCallbackSub while instantiating these classes, and then call **Start()** or **Connect()**, i.e.:

```
Dim _Server As New tcpCommServer(AddressOf YourCallbackSub)
_Server.Start(60000)
```

Or:

```
Dim _Client As New tcpCommClient(AddressOf YourCallbackSub)
_Client.Connect("192.168.1.1", 60000)
```

YourCallbackSub has to have the right signature - you can see the right way to do it in the example project. You can also specify the max bps while instantiating the server class. The default value is 9MBps.

To send a byte array, use **sendBytes()**. To get a file from the server, there is a **GetFile()** method. Simply supply it with the path of the file on the server (a path local to the server). To send a file to the server, there's a **SendFile()** method. Both **GetFile** and **SendFile** are handled by the classes without any other code needed by you - but if you'd like to track the progress of the incoming or outgoing file, you can poll the **GetPercentOfFileReceived** and **GetPercentOfFileSent** methods.

These classes communicate with each other, and with you using a limited protocol language. You will be notified in your callback when your bytes have been sent (using **sendBytes**), when your file has completed downloading or uploading, if a file transfer has been aborted, and if you receive an error either locally or from the server. You will receive these messages from your client or

server on channel 255. This is a quick look at these messages, and how I handled them in the example project's client form's callback sub:

```
Public Sub UpdateUI(ByVal bytes() As Byte, ByVal dataChannel As Integer)

    If Me.InvokeRequired() Then
        ' InvokeRequired: We're running on the background thread. Invoke the delegate.
        Me.Invoke(_Client.ClientCallbackObject, bytes, dataChannel)
    Else
        ' We're on the main UI thread now.
        Dim dontReport As Boolean = False

        If dataChannel < 251 Then
            Me.ListBox1.Items.Add(BytesToString(bytes))
        ElseIf dataChannel = 255 Then
            Dim msg As String = BytesToString(bytes)
            Dim tmp As String = ""

            ' Display info about the incoming file:
            If msg.Length > 15 Then tmp = msg.Substring(0, 15)
            If tmp = "Receiving file:" Then
                gbGetFileProgress.Text = "Receiving: " & _Client.GetIncomingFileName
                dontReport = True
            End If

            ' Display info about the outgoing file:
            If msg.Length > 13 Then tmp = msg.Substring(0, 13)
            If tmp = "Sending file:" Then
                gbSendFileProgress.Text = "Sending: " & _Client.GetOutgoingFileName
                dontReport = True
            End If

            ' The file being sent to the client is complete.
            If msg = "->Done" Then
                gbGetFileProgress.Text = "File->Client: Transfer complete."
                btGetFile.Text = "Get File"
                dontReport = True
            End If

            ' The file being sent to the server is complete.
            If msg = "<-Done" Then
                gbSendFileProgress.Text = "File->Server: Transfer complete."
                btSendFile.Text = "Send File"
                dontReport = True
            End If

            ' The file transfer to the client has been aborted.
            If msg = "->Aborted." Then
                gbGetFileProgress.Text = "File->Client: Transfer aborted."
                dontReport = True
            End If

            ' The file transfer to the server has been aborted.
            If msg = "<-Aborted." Then
                gbSendFileProgress.Text = "File->Server: Transfer aborted."
                dontReport = True
            End If

            ' _Client as finished sending the bytes you put into sendBytes()
            If msg = "UBS" Then ' User Bytes Sent.
                btSendText.Enabled = True
                dontReport = True
            End If
        End If
    End If
End Sub
```

```

    ' We have an error message. Could be local, or from the server.
    If msg.Length > 4 Then tmp = msg.Substring(0, 5)
    If tmp = "ERR: " Then
        Dim msgParts() As String
        msgParts = Split(msg, ": ")
        MsgBox("'" & msgParts(1), MsgBoxStyle.Critical, "Test Tcp Communications App")
        dontReport = True
    End If

    ' Display all other messages in the status strip.
    If Not dontReport Then Me.ToolStripStatusLabel1.Text = BytesToString(bytes)
End If
End If

End Sub

```

Sendbytes will accept any size byte array you hand it. It can only send **blockSize** bytes at a time though, so if you hand it a very large byte array, **sendBytes** will block until it's done sending it. You will want to wait until you receive the "UBS" notice before handing **sendBytes** more data.

These classes will work the most efficiently if you only try to send a maximum of **blockSize** bytes at a time. **BlockSize** will be different depending on your current throttle speed, and the server will change the client's **blockSize** immediately after connection. You can get the current **blockSize** by calling the **.GetBlocksize()** method.

Throttling

Throttling works along a 4K boundary if you set the bandwidth to be less than 1 meg, so you can use the traditionally expected bandwidth limits - i.e., 64K, 96K, 128K, 256K, etc. After one meg, it's along a 5K boundary - so you can set it to more intuitive values: 2.5 meg, 5 meg, 9 meg, etc. The server counts all bytes coming in and going out, and checks to make sure we're not processing more bytes than we should 4 times a second. For those who are interested, this is what the code looks like:

```

' Throttle network Mbps...
bandwidthUsedThisSecond = session.bytesSentThisSecond + session.bytesRecievedThisSecond
If bandwidthTimer.AddMilliseconds(250) >= Now And bandwidthUsedThisSecond >= (Mbps / 4) Then
    While bandwidthTimer.AddMilliseconds(250) > Now
        Thread.Sleep(1)
    End While
End If
If bandwidthTimer.AddMilliseconds(250) <= Now Then
    bandwidthTimer = Now
    session.bytesRecievedThisSecond = 0
    session.bytesSentThisSecond = 0
    bandwidthUsedThisSecond = 0
End If

```

Throttling will be important for you because these classes will do as much work as you let them - as fast as they can. On my dev machine, I was able to achieve transfer speeds of almost 300 Mbps copying one large file from the server to the client. But the cost of this was the client and the server's background threads using 100% of their respective CPU core's resources - not a good scenario for a server. On my dev machine, 9MBps produced almost no perceptible CPU usage. In a production environment, I'm sure you will want to set this even lower. Ideally, you will want to set the throttling when you instantiate your server.

```

Dim _Server As New tcpCommServer(AddressOf YourCallbackSub, 9000000)

```

You can also set the throttled bps using the **ThrottleNetworkBps()** method.

The CpuMonitor Class

Included in this project, you'll find the **CpuMonitor** class. This class is used to monitor the CPU usage of a *thread*. When running the example project, you will see some reporting of CPU usage in the status strip. This is the usage of the client's background thread - the thread actually doing the work of communicating with the server.

Points of Interest

Following is a list of changes available in version **4.08** of this library.

- Configured the example project so that large arrays can be sent to the server on channel 100 as well as received from the server on that channel. To send the example large array to the server on channel 100, click Send in the client when the send text box is empty.

Following is a list of changes available in version **4.07** of this library.

- Redesigned the CPU idle throttling / disconnection detection system so that disconnections are more reliably detected. Single byte keep alive packets are now sent from the client and the server every two seconds during idle time.

Following is a list of changes available in version **4.06.1** of this library.

- **Bugfix:** The client now polls **TcpClient.Client.Connected** as part of its normal connection verification testing. This is to ensure that the client is made aware of loss of connection while it is sending data.

Following is a list of changes available in version **4.05** of this library.

- The client will now disconnect after failing to receive a **keepalive** packet from the server for longer than 7 seconds (the server sends them every 5 seconds).

Following is a list of changes available in version **4.04** of this library.

- Added the ability to specify which **ipaddress** you would like the server to listen on in your system, or select **0.0.0.0** for **IPAddress.Any** via a combo box on the server form.

Following is a list of changes available in version **4.03** of this library.

- Added locking and synchronization code to **GetFile()** and **SendFile()**. Both are now thread safe and extremely performant when sending or receiving many small files.

Following is a list of changes available in version **4.02** of this library.

- **Bugfix:** Corrected an issue where Mbps wasn't being reported correctly when sending a file.

Following is a list of changes available in version **4.01** of this library.

- **Bugfix:** The client was using excessive CPU resources (the tcp library wasn't idling while inactive). Found by [skdia15](#).

Following is a list of changes available in version **4.0** of this library.

- This version does NOT use a synchronization byte. This change was to increase performance during file transfers.
- The throttling system can now be disabled, by entering a **0** in the server's constructor instead of a max Mbps value.
- **Bugfix:** A bug that was causing the client to disconnect occasionally was found and resolved (attempting to close a **filestream** that had been disposed).
- Because this version does not use the synchronization byte any longer, it is NOT compatible with previous versions.

Following is a list of changes available in version **3.963** of this library.

- **Bugfix:** The **GetSendQueueSize()** function was not reporting correctly - found by [Bishop44](#)

Following is a list of changes available in version **3.962** of this library.

- **Bugfix:** Resolved a thread synchronization issue involving the write queue. I have removed the write queue for now.

Following is a list of changes available in version **3.961** of this library.

- **Bugfix:** Resolved a thread synchronization issue in the new file transfer code.

Following is a list of changes available in version **3.96** of this library.

- Implemented an overhaul of the file transfer system. These changes were made to ensure robust and performant transfers of files to and from the server. An example of how to send multiple files and folders, view the server's remote file system, and even be afforded some limited management functionality using this library will follow shortly (in another article).
- **Bugfix:** The crash in the `GetPercentOfFile...()` has been resolved.
- **Bugfix:** The hang some people experienced while attempting to transfer multiple files has been resolved.
- The 10k blocksize issue has been resolved. `Blocksize` will now be set to `62500` unless the throttling value is set lower than that.
- A `concurrentQueue` has been implemented in the client and server classes to replace the `AsyncUnbufWriter` class, improving performance.
- Four new delegates have been added for people to subscribe to: `incomingFileProgress`, `outgoingFileProgress`, `incomingFileComplete` and `outgoingFileComplete`. Have a look at the code in the client form to see how to use them.
- `ReceivingAFile()` and `SendingAFile()` functions have been added to the client class, and the server session class. They both return a boolean value, and return `false` when your file is complete.

Following is a list of changes available in version **3.95** of this library.

- Added an "Active file transfers" `listview` to the server form. All file transfers from every connected client can be monitored here, and file transfers that are aborted or complete are removed. Code needed to be borrowed from the client and added to the server session class for this to work properly.

Following is a list of changes available in version **3.94** of this library.

- Fixed a bug in `client.Connect()` that was preventing VS 2015 from compiling the example project.

Following is a list of changes available in version **3.93** of this library.

- Introduced a new UI update pattern that is more efficient.

Following is a list of changes available in version **3.92** of this library.

- **Bugfix:** The server was not updating the `blocksize` in clients after the first connected. Reported by [Bishop44](#).

Following is a list of changes available in version **3.91** of this library.

- **Bugfix:** Corrected threading problem with the Large Array Transfer Helper class: Crashes / LAT failures were sometimes occurring due to list items being added / removed by multiple threads.
- **Patch:** Auto-reconnecting after connect failure: Clients configured to auto-reconnect were attempting to auto-reconnect after a connection failure. This was unintended behavior.
- **Patch:** Clients not correctly displaying connection status after clicking the disconnect button while attempting to auto-reconnect: Also resolved in this version.

Following is a list of changes available in version **3.9** of this library.

- **Patch:** Added options to `client.close()` and `sesison.close()`: The default behavior is now to wait until messages in the `send` queue have been sent before shutting down.

Following is a list of changes available in version **3.8** of this library.

- **Bugfix:** Patch for an bug accidentally introduced in V3.7: Clients deliberately disconnected from the server now correctly report it. V3.8 appears to function correctly in all areas, and unless someone reports a bug or I discover something else, this should be the last update for a while.

Following is a list of changes available in version **3.7** of this library.

- **Bugfix:** I throttled the new connection speed down in the server. This server can now accept 100 new connections per second. Of course, if more than 100 attempt to connect in a second, they will be able to. They will just have to wait a little longer. This is to provide time for the .NET garbage collector to do its job.
- **New Functionality:** I also added some new checkboxes to the example project so you can test automatic reconnects and server **machineID** enforcement (or not).

Following is a list of changes available in version **3.6** of this library.

- **Bugfix:** Sessions that were disconnected from a server for connecting with a blank **MachineID** were getting an empty error message before disconnect. I also did much more work ensuring that sessions that connect with duplicate machine ids, or no id are rejected properly. Clients trying to connect to a server with duplicate **machineIDs**, or a blank **machineID** receive a rejection notice before the **connect()** function returns, and as such, rejected connection attempts will cause **connect()** to return **false**. An error message with an explanation will also be received in your callback. A Connection success notice will also be received before **connect()** returns, which means before **connect()** passes control back to you, your session is already visible in the server's session collection. Disconnected sessions have their **sessionIDs** replaced with an empty **string**, and are marked as not validated (checked to ensure uniqueness). As such, they are not visible in the session collection at all.

Following is a list of changes available in version **3.5** of this library.

- **Bugfix:** A memory leak, sessions bumping off other sessions, duplicate sessions, and other issues with the session management system have been resolved. - Reported by Earthed (Thank you).
- **New Functionality:** Automatic session reconnecting during disconnect events. While initializing a client, you can specify an auto-reconnect duration. The default is 30 seconds.
- **New Functionality:** The server can now be set to enforce unique IDs. This is the default behavior, but you can specify not to in the server init if you would like to use the machine IDs for something else.

Following is a list of changes available in version **3.1** of this library.

- **Bugfix:** **Utilities.LargeArrayTransferHelper.VerifySignature()** was returning an object when it should have been returning a Boolean.
- **Graceful disconnect** - Previously, the client would close without notifying the server. This would sometimes leave the session running in the server, leading to increased CPU and memory usage over time.
- **BugFix:** Deadlock while attempting to send bytes from the callback. In version 2 and below, using **SendBytes()** from the callback would sometimes result in a deadlock condition. This has been resolved by the introduction of send and receive packet queuing in both the client and the server classes.
- **New Functionality:** Method **GetSendQueueSize()** added. Use this to monitor how fast the messages you drop into the queue are being sent. If this number rises, then you are queuing messages faster than you can send them. This is a client only method.
- **New Functionality:** The library has been rolled into a **.dll** project. This was to make it convenient for programmers (ahem... me) using other .NET languages to have access to the functionality in these libraries.
- **New Functionality:** both the client and its server session now track a **'machineID'**. This is a **string**, and can be anything you like. The idea is that it can be used as a unique machine identifier, above and beyond the **sessionId**. Use the **SetMachineID()** and **GetMachineID()** client methods for this. Also, you can now retrieve a server session using a **machineID** (using **TcpComm.server.GetSession(machineid)**).
- **BugFix:** Reuse of sessionIDs. Previously, the server would add sessions to an **Arraylist**. Disconnected sessions were left in the list, and new sessions would get an ever increasing ID number as they were added. It was feasible (though unlikely), that very busy servers could eventually crash because they'd reached the upper limit of the **sessionId** integer. Version 3 of this library avoids this issue by reusing **sessionIDs**.
- **New Functionality:** The server's **sessionCollection** object and **SessionCommunication** class are now exposed. The helper functions: **GetSessionCollection()**, **GetSession(int)** and **GetSession(string)** have been added to assist in server management. Please look at the example project to see how this is done.
- **New Functionality:** **SendText()** added. This is exactly what it looks like - a convenience function used to send simple text messages. Use **SendBytes()** for jobs other than text messages.
- **New Functionality:** **client.Connect()** now resolves hostnames and domain names via DNS. You no longer need to enter an IP address in the **client.connect()** method - it can be anything resolvable by DNS.

- **BugFix:** Addresses other than valid IPv4 addresses are sometimes incorrectly returned. Reported by [ThaviousXVII](#).
- **BugFix:** Files transferred are sometimes not released (closed) by the receiver. Reported by [Rich Frasier](#).
- **BugFix:** The server can now initiate a file transfer to a client. Reported by [Kasram](#) and others.
- **New Functionality:** A convenience class used to transfer large arrays easily. Use the [Utilities.LargeArrayTransferHelper](#) class. See the example project for usage and notes.

Lastly, I have to say that I really appreciate the bug reports. I know we all have busy, hectic lives these days (I know I do), so thanks for taking the time to post that stuff here.

History

- 12/30/11: Corrected a small problem with protocol reporting in [tcpCommServer](#). Added notification for when [sendBytes\(\)](#) has completed sending data.
- 6/19/12: Resolved file IO issues. Added the [AsyncUnbuffWriter](#) class to the example project. Resolved all warnings and rebuilt the project using Option Strict and Option Explicit.
- 4/14/14 - V3 of the library uploaded. Please see above for details.


License

This article, along with any associated source code and files, is licensed under [The Code Project Open License \(CPOLE\)](#)

About the Author




pdoxtader

President Doxtader Industries LLC
United States 

I've been in IT for the last 20 years in one capacity or another - always as either a network engineer or a developer... or both. At the moment I have an IT consultancy in Long Island, NY offering software development and network engineer services.

Comments and Discussions

 **842 messages** have been posted for this article Visit <https://www.codeproject.com/Articles/307315/Reusable-Multithreaded-TCP-IP-Client-and-Server-CI> to post and view comments on this article, or click [here](#) to get a print view with messages.

[Permalink](#)
[Advertise](#)
[Privacy](#)
[Cookies](#)
[Terms of Use](#)

Article Copyright 2011 by pdoxtader
Everything else Copyright © [CodeProject](#),
1999-2019

Web05 2.8.191108.1

