Phone Mic Live!

CSE Mini-Project 6th Semester

A Windows Phone Application and a server side application to transform the phone into a wireless microphone.

SOUHAM BISWAS

Table of Contents

INTRODUCTION

Overview –

This report is a documentation of the workings of the Phone Mic Live! Application. This includes the black box test scenarios coupled alongwith the white box test scenarios.

Motivation and Background –

This project is basically a low-cost alternative to having a proper microphone system for making announcements etc. in a populated environment.

It involves a package of 2 applications; One running on the phone, and the other application to run on the system connected to the audio equipment.

The sound entering the phone is played through the audio equipment connected to the system to which the phone is linked.

ACKNOWLEDGEMENT

I would like to thank Prof. R.R. Tewari for his invaluable guidance provided throughout the project development.

Also, this project would have been impossible without the continuous support received from the Microsoft Developer Network (MSDN).

Lastly, but not the least, I thank everyone else involved directly or indirectly with the development of this software, as this page is too short to list everyone.

CERTIFICATE

UNIVERSITY OF ALLAHABAD
QUOT RAMI TOT ARBORES

This is to certify that Mr. Souham Biswas has successfully prepared and completed the project under my direct and close supervision and that this is a bonafide piece of work done by him.

Class: B.Tech , VIth semester

Branch: Computer Science Engineering

Academic Year: 2013-14

Institution Name: J.K. Institute of Applied Physics & Technology

Signature of Examiner: ______________________

Date: ______________________

SOFTWARE PRE-REQUISITES

- **.NET Runtime 4.0 (on server side)**

  The application has been developed in C#.
  The fact that it is .NET 4.0 based allows it to run on most Windows based machines, as .NET 4.0 is shipped by default alongwith all Windows Operating Systems.

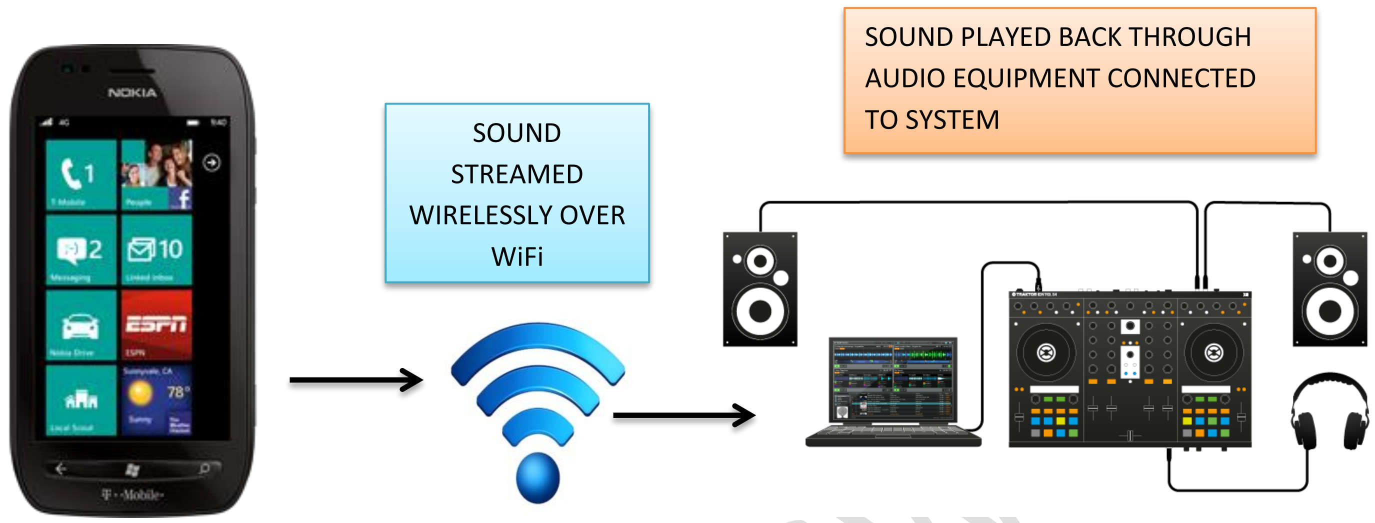
- **WiFi hardware (on server side)**

  The audio captured by the phone is streamed over a live link maintained between the phone and the target system via the WiFi radio hardware.

- **Windows Phone OS 7.1 (on phone)**

  The phone side application has been developed for the Windows Phone 7.1 Mango OS Environment.
  The phone application runs in a .NET CF(Compact Framework) environment, which is a stripped version of the original .NET framework optimized for devices with memory and computational constraints.

SOFTWARE OVERVIEW


SOUND STREAMED WIRELESSLY OVER WiFi
SOUND PLAYED BACK THROUGH AUDIO EQUIPMENT CONNECTED TO SYSTEM

This application transforms an ordinary Windows Phone into a full fledged wireless microphone.

This involves an app running on the phone, and an app running on the system connected to the speakers.

An ad-hoc WiFi network hotspot is established by the app on the system to which the phone connects. A live TCP connection is maintained between the phone and the system over which all the audio entering the phone is streamed to the system and played by the speaker system in real-time.

APPLICATION FEATURES

- Large Wireless Range –
  Most wireless mics don't have a range beyond 10 meters. But since this is dependent on the WiFi radio, which usually extends much beyond 10 meters, hence gives a huge range bonus over conventional wireless microphone systems.

- Mobility and versatility –
  The Windows Phone application is scalable across all Windows Phone devices, which allows it to run on any Windows Phone device. Moreover, it can be easily ported to other platforms like Android, using the Mono (Xamarin) framework, as this is written in C# language.

- Compatibility across Windows Platforms –
  The server side application was developed in the C# programming language which was developed by Microsoft, and requires a minimum of .NET 4.0 runtime to run. Now since most Windows Operating systems are already shipped with .NET 4.0 Framework, hence it renders the server application compatible across most Windows platforms.

- Cheap and powerful alternative –
  This can be easily used in place of an actual wireless mic system at no extra cost. Hence, it is pretty much a cheap and powerful alternative.

TECHNOLOGIES AND SOFTWARES USED

<u>C# Programming Language –</u>

C# is a .NET based object oriented programming language which is analogous to Java, but is much more powerful and object oriented.

It was developed by Microsoft as a response to rising utility for internet and cloud based applications.

C# applications run in a virtual machine called the CLR or Common Language Runtime, which allows for automatic memory management and native machine code optimization.

The CLR allows any .NET based language (VB, F#, etc.) to run in a single common environment, which hence allows for code sharing.

C# code on compilation is converted to MSIL or Microsoft Intermediate Language, instead of native machine code as in the case of C or C++. Then, when the application is run, only those modules in use, are conditionally compiled to native machine code before execution. This compilation technique is popularly known as JIT or Just In Time compilation and results in a smaller memory footprint.

<u>Microsoft XNA Framework–</u>

This is a framework developed by Microsoft to work with media elements pertaining to audio or video. It usually finds prime usage in game development. Elements of this framework have been employed in this project to perform basic audio processing tasks

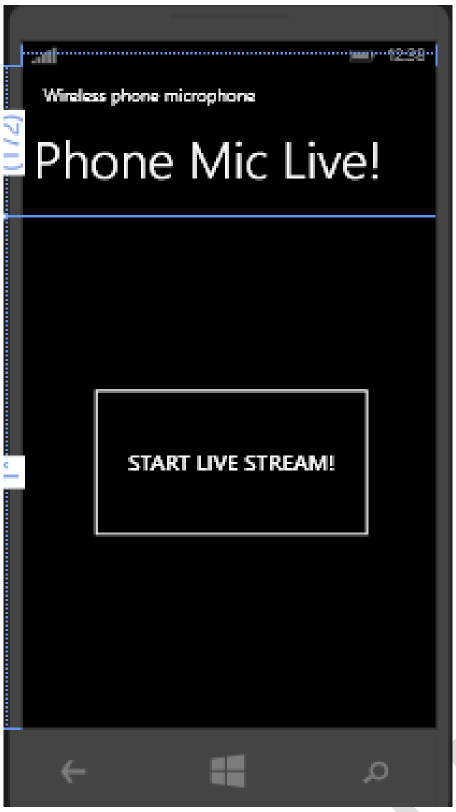
.

NAudio Library –

NAudio is an open source .NET audio and MIDI library, containing dozens of useful audio related classes intended to speed development of audio related utilities in .NET. This has been used in this project to process and play the stream of raw PCM audio stream sent by the phone.

SOFTWARE WORKINGS

Phone Application



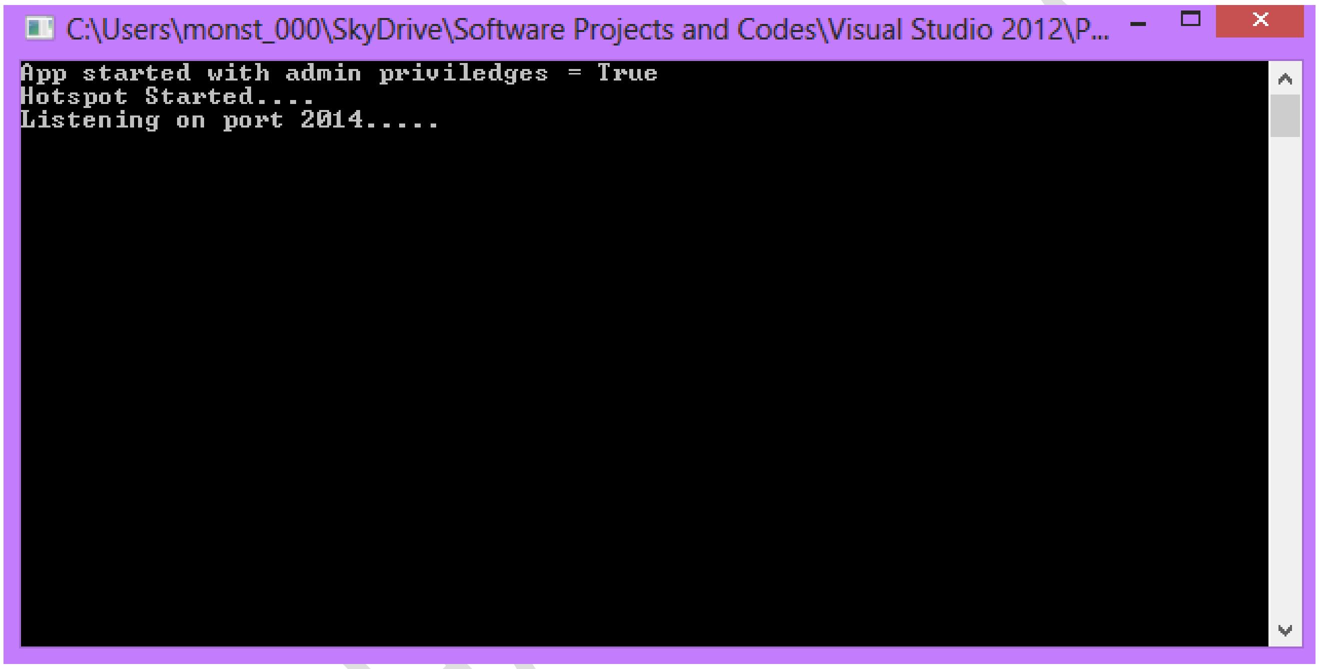
This is a screenshot of the corresponding Windows Phone application. Upon tapping the “START LIVE STREAM!” button, the phone starts to stream audio incident on it’s microphone over the wireless link to the target system.

The audio is captured in raw PCM (Pulse Code Modulated) format, and is parsed into byte arrays which are stored temporarily.

The audio is captured into buffers which store audio information upto 100 ms, and then are serially transmitted in packets as raw PCM data to the server system.

The phone must be connected to the Wifi network hotspot started by the server side application prior to starting the stream for successful operation.

Server Application



```
C:\Users\monst_000\SkyDrive\Software Projects and Codes\Visual Studio 2012\P...
App started with admin priviledges = True
Hotspot Started....
Listening on port 2014.....
```

The server side application is basically a console application which initializes a WiFi ad-hoc network hotspot to which the phone connects and streams the audio.

This is done by automatically passing commands to the native command prompt tool in Windows Operating Systems.

Once the network is up, the application starts to listen for incoming audio packets over port 2014.

As and when a packet is received, it is immediately played back in it's raw, PCM format. This has been achieved with the help of the NAudio Library.

The protocol through which the communication takes place is TCP.

Expansion Scope-

The server side application can be made into a nice GUI, and the overall application maybe expanded to allow for recording, storing and adding digital effects to incoming audio.

Moreover, a wireless camera application may be built on this framework.

CODE

Phone Application

MainPage.xaml

```
<phone:PhoneApplicationPage
    x:Class="WifiPhoneApp1.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:phone="clr-namespace:Microsoft.Phone.Controls;assembly=Microsoft.Phone"
    xmlns:shell="clr-namespace:Microsoft.Phone.Shell;assembly=Microsoft.Phone"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d" d:DesignWidth="480" d:DesignHeight="768"
    FontFamily="{StaticResource PhoneFontFamilyNormal}"
    FontSize="{StaticResource PhoneFontSizeNormal}"
    Foreground="{StaticResource PhoneForegroundBrush}"
    SupportedOrientations="Portrait" Orientation="Portrait"
    shell:SystemTray.IsVisible="True">

    <!--LayoutRoot is the root grid where all page content is placed-->
    <Grid x:Name="LayoutRoot" Background="Transparent">
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="*"/>
        </Grid.RowDefinitions>

        <!--TitlePanel contains the name of the application and page title-->
        <StackPanel x:Name="TitlePanel" Grid.Row="0" Margin="12,17,0,28">
            <TextBlock x:Name="ApplicationTitle" Style="{StaticResource
PhoneTextNormalStyle}">
                  <Run Text="Wireless phone microphone"/>
                  <LineBreak/>
                  <Run/>
            </TextBlock>
            <TextBlock x:Name="PageTitle" Margin="0,-7,0,0" Style="{StaticResource
PhoneTextTitle1Style}" FontSize="60">
                  <Run Text="Phone"/>
                  <Run Text=" "/>
                  <Run Text="Mic Live!"/>
            </TextBlock>
        </StackPanel>

        <!--ContentPanel - place additional content here-->
        <Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
```

```
            <Button Content="START LIVE STREAM!" HorizontalAlignment="Left"
Margin="60,190,0,0" VerticalAlignment="Top" Width="342" Height="194"
Click="Button_Click_1"/>
        </Grid>
    </Grid>

</phone:PhoneApplicationPage>
```

MainPage.xaml.cs

```
using Microsoft.Phone.Controls;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using System;
using System.IO;
using System.Windows;
using System.Windows.Threading;

namespace WifiPhoneApp1
{
    public partial class MainPage : PhoneApplicationPage
    {
        SocketClient sc = new SocketClient();
        string testIP = "192.168.173.1";
        int testPort = 2014;

        Microphone mic = Microphone.Default;
        byte[] buffer;
        MemoryStream mStream = new MemoryStream();
        SoundEffect sound;
        bool Connected = false;

        // Constructor
        public MainPage()
        {
            InitializeComponent();
            DispatcherTimer dt = new DispatcherTimer();
            dt.Interval = TimeSpan.FromMilliseconds(33);
            dt.Tick += delegate { try { FrameworkDispatcher.Update(); } catch { }
};
            dt.Start();
            mic.BufferReady += new EventHandler<EventArgs>(mic_BufferReady);
        }

        private void sendData(byte[] data)
        {
            sc.Connect(testIP, testPort);
            sc.Send(data);
        }

        private void mic_BufferReady(object sender, EventArgs e)
        {
            mic.GetData(buffer);
            sendData(buffer);
            mStream.Write(buffer, 0, buffer.Length);
        }

        private void Button_Click_1(object sender, RoutedEventArgs e)
        {
            mic.BufferDuration = TimeSpan.FromMilliseconds(100);
            buffer = new byte[mic.GetSampleSizeInBytes(mic.BufferDuration)];
            mic.Start();
        }

    }

}
```

SocketClient.cs

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

namespace WifiPhoneApp1
{
    public class SocketClient
    {

        Socket _socket = null;

        static ManualResetEvent _clientDone = new ManualResetEvent(false);

        const int TIMEOUT_MILLISECONDS = 5000;

        const int MAX_BUFFER_SIZE = 2048;

        /// <summary>
        /// Attempt a TCP socket connection to the given host over the given port
        /// </summary>
        /// <param name="hostName">The name of the host</param>
        /// <param name="portNumber">The port number to connect</param>
        /// <returns>A string representing the result of this connection
attempt</returns>
        public string Connect(string hostName, int portNumber)
        {
            string result = string.Empty;

            // Create DnsEndPoint. The hostName and port are passed in to this
method.
            DnsEndPoint hostEntry = new DnsEndPoint(hostName, portNumber);
            // Create a stream-based, TCP socket using the InterNetwork Address
Family.
            _socket = new Socket(AddressFamily.InterNetwork, SocketType.Stream,
ProtocolType.Tcp);

            // Create a SocketAsyncEventArgs object to be used in the connection
request
            SocketAsyncEventArgs socketEventArg = new SocketAsyncEventArgs();
            socketEventArg.RemoteEndPoint = hostEntry;

            // Inline event handler for the Completed event.
            // Note: This event handler was implemented inline in order to make
this method self-contained.
            socketEventArg.Completed += new
EventHandler<SocketAsyncEventArgs>(delegate(object s, SocketAsyncEventArgs e)
            {
                // Retrieve the result of this request
                result = e.SocketError.ToString();

                // Signal that the request is complete, unblocking the UI thread
                _clientDone.Set();
            });

            // Sets the state of the event to nonsignaled, causing threads to
block
            _clientDone.Reset();

            // Make an asynchronous Connect request over the socket
            _socket.ConnectAsync(socketEventArg);


            // Block the UI thread for a maximum of TIMEOUT_MILLISECONDS
milliseconds.
            // If no response comes back within this time then proceed
            _clientDone.WaitOne(TIMEOUT_MILLISECONDS);

            return result;
        }

        /// <summary>
        /// Send the given string data to the server using the established
connection
        /// </summary>
        /// <param name="data">The data to send to the server</param>
        /// <returns>The result of the Send request</returns>
        public string Send(string data)
        {
            return sendMethod(data, new byte[0], true);
        }

        /// <summary>
        /// Send the given raw data (in a byte stream) to the server using the
established connection
        /// </summary>
        /// <param name="payload">The raw byte data to send to the server</param>
        /// <returns>The result of the Send request</returns>
        public string Send(byte[] payload)
        {
            return sendMethod(String.Empty, payload, false);
        }

        private string sendMethod(string data, byte[] inputPayload, bool
stringData)
        {
            string response = "Operation Timeout";

            // We are re-using the _socket object initialized in the Connect
method
            if (_socket != null)
            {

                // Create SocketAsyncEventArgs context object
                SocketAsyncEventArgs socketEventArg = new SocketAsyncEventArgs();

                // Set properties on context object
                socketEventArg.RemoteEndPoint = _socket.RemoteEndPoint;
////////////////
                socketEventArg.UserToken = null;

                // Inline event handler for the Completed event.
                // Note: This event handler was implemented inline in order
                // to make this method self-contained.
                socketEventArg.Completed += new
EventHandler<SocketAsyncEventArgs>(delegate(object s, SocketAsyncEventArgs e)
                {
                    response = e.SocketError.ToString();

                    // Unblock the UI thread
                    _clientDone.Set(); ////////////////
                });

                // Add the data to be sent into the buffer
                if (stringData)
                    inputPayload = Encoding.UTF8.GetBytes(data);
                socketEventArg.SetBuffer(inputPayload, 0, inputPayload.Length);

                // Sets the state of the event to nonsignaled, causing threads to
block
                _clientDone.Reset();

                // Make an asynchronous Send request over the socket
                _socket.SendAsync(socketEventArg);

                // Block the UI thread for a maximum of TIMEOUT_MILLISECONDS
milliseconds.
                // If no response comes back within this time then proceed
                _clientDone.WaitOne(TIMEOUT_MILLISECONDS);
            }
            else
            {
                response = "Socket is not initialized";
            }

            return response;
        }
    }
}
```

Server Side Application

Program.cs

```
using NAudio.Wave;
using System;
using System.Diagnostics;
using System.IO;
using System.Security.Principal;
using TCPConnector;


namespace WifiTestCommApp
{
    class Program
    {
        static int PORT = 2014;

        static void Main(string[] args)
        {
            if (!IsAdmin())
                RestartElevated();
            Console.WriteLine("App started with admin priviledges = " + 
IsAdmin());
            WifiHotspotComm.Start_Hotspot("PhoneMicLive!", "89483653690", true);
            Console.WriteLine("Hotspot Started....");
            TCPDataReader TCPRdr = new TCPDataReader(PORT);
            TCPRdr.ReceiveBufferSize = 3200;
            TCPRdr.StartListener();


            Console.WriteLine("Listening on port " + PORT + ".....");
            MemoryStream s;
            while (true)
            {
                s = new MemoryStream(TCPRdr.GetReceivedByteData());
                var waveFormat = new WaveFormat(16000, 16, 1); // must match the 
waveformat of the raw audio
                var waveOut = new WaveOut();
                var rawSource = new RawSourceWaveStream(s, waveFormat);
                waveOut.Init(rawSource);
                waveOut.Play();
            }
            Console.Read();
            WifiHotspotComm.Start_Hotspot(null, null, false);
        }

        public static bool IsAdmin()
        {
            WindowsIdentity id = WindowsIdentity.GetCurrent();
            WindowsPrincipal p = new WindowsPrincipal(id);
            return p.IsInRole(WindowsBuiltInRole.Administrator);
        }

        public static void RestartElevated()
        {
            ProcessStartInfo startInfo = new ProcessStartInfo();
            startInfo.UseShellExecute = true;
            startInfo.CreateNoWindow = true;
            startInfo.WorkingDirectory = Environment.CurrentDirectory;
            startInfo.FileName = System.AppDomain.CurrentDomain.FriendlyName;
            //startInfo.FileName =
System.Windows.Forms.Application.ExecutablePath;
            startInfo.Verb = "runas";
            try
            {
                Process p = Process.Start(startInfo);
            }
            catch (Exception e)
            {
                Console.WriteLine(e.ToString());
            }
            Environment.Exit(0);
            //System.Windows.Forms.Application.Exit();
        }
    }
}
```

WifiHotspotComm.cs

```
using System;
using System.Diagnostics;

namespace WifiTestCommApp
{
    class WifiHotspotComm
    {
        public static void Start_Hotspot(string ssid, string key, bool status)
        {
            ProcessStartInfo processStartInfo = new ProcessStartInfo("cmd.exe");
            processStartInfo.RedirectStandardInput = true;
            processStartInfo.RedirectStandardOutput = true;
            processStartInfo.CreateNoWindow = true;
            processStartInfo.UseShellExecute = false;
            Process process = Process.Start(processStartInfo);

            if (process != null)
            {
                if (status)
                {
                    process.StandardInput.WriteLine("netsh wlan set hostednetwork 
mode=allow ssid=" + ssid + " key=" + key);
                    process.StartInfo.RedirectStandardOutput = true;
                    process.StandardInput.WriteLine("netsh wlan start hosted 
network");
                    process.StandardInput.Close();
                }
                else
                {
                    process.StandardInput.WriteLine("netsh wlan stop 
hostednetwork");
                    process.StandardInput.Close();
                }
            }
            //Console.WriteLine(process.StandardOutput.ReadToEnd());
        }
    }
}
```

TCPConnector.cs

```
using System.Net;
using System.Net.Sockets;
using System.Text;

namespace TCPConnector
{
    public class TCPDataReader
    {

        #region Class Properties and Fields

        /// <summary>
        /// Port over which data will be exchanged.
        /// </summary>
        public int PORT_NO
        {
            get { return privatePORT_NO; }
            set { initializeWithPORT_NUMBER(value); }
        }

        /// <summary>
        /// IP Address of incoming data source.
        /// </summary>
        public string IP_ADDRESS
        {
            get { return privatelocalAdd.ToString(); }
            set { initializeMethod(value, privatePORT_NO); }
        }

        /// <summary>
        /// Size of data receiving buffer size. Default = 8192
        /// </summary>
        public int ReceiveBufferSize
        {
            get { return client.ReceiveBufferSize; }
            set { privateReceiveBufferSize = value; }
        }

        private int privateReceiveBufferSize = 8192;

        /// <summary>
        /// String data received.
        /// </summary>
        public string ReceivedStringData
        {
            get { return stringDataReceived; }
        }

        /// <summary>
        /// Byte stream data received.
        /// </summary>
        public byte[] ReceivedByteData
        {
            get { return buffer; }
        }

        private int privatePORT_NO;
        private IPAddress privatelocalAdd = IPAddress.Any;
        private TcpListener listener;
        private TcpClient client;
        private NetworkStream nwStream;
        private int bytesRead;

        private byte[] buffer;
        private string stringDataReceived;

        #endregion

        #region Class Constructors

        /// <summary>
        /// Constructor accepting IP Address and Port number over which data will
be exchanged.
        /// </summary>
        /// <param name="IP_ADDRESS">IP Address of incoming data source.</param>
        /// <param name="PORT_NUMBER">Port over which data will be
received.</param>
        public TCPDataReader(string IP_ADDRESS, int PORT_NUMBER)
        {
            initializeMethod(IP_ADDRESS, PORT_NUMBER);
        }

        /// <summary>
        /// Constructor accepting only Port Number over which data will be
exchanged.
```

```
        /// Data will be exchanged over ANY IP Address using this port number if
IP_ADDRESS property is left blank.
        /// </summary>
        /// <param name="PORT_NUMBER">Port over which data will be
received.</param>
        public TCPDataReader(int PORT_NUMBER)
        {
            initializeWithPORT_NUMBER(PORT_NUMBER);
        }

        #endregion

        #region Initialization Methods

        /// <summary>
        /// Used to initialize object taking both IP Address and Port Number as
input.
        /// </summary>
        /// <param name="IP_ADDRESS">IP Address from which to accept data.</param>
        /// <param name="PORT_NUMBER">Port over which data will be
exchanged.</param>
        private void initializeMethod(string IP_ADDRESS, int PORT_NUMBER)
        {
            privatelocalAdd = IPAddress.Parse(IP_ADDRESS);
            initializeWithPORT_NUMBER(PORT_NUMBER);
        }

        /// <summary>
        /// Used to initialize object taking only port number. Data will be
accepted from any IP Address.
        /// </summary>
        /// <param name="PORT_NUMBER">Port over which data will be
exchanged.</param>
        private void initializeWithPORT_NUMBER(int PORT_NUMBER)
        {
            this.privatePORT_NO = PORT_NUMBER;
            listener = new TcpListener(this.privatelocalAdd, this.privatePORT_NO);
        }

        #endregion

        #region Data Procuring Methods

        /// <summary>
        /// Starts the TCP Listener.
        /// </summary>
        public void StartListener()
        {
            listener.Start();
        }

        /// <summary>
        /// Gets the data received over the specified IP Address and Port number
and updates the ReceivedData property.
        /// </summary>
        /// <returns>Data Received over the given port.</returns>
        public string GetReceivedStringData()
        {
            GetReceivedByteData();

            //---convert the data received into a string---
            stringDataReceived = Encoding.ASCII.GetString(buffer, 0, bytesRead);
            return stringDataReceived;
        }

        /// <summary>
        /// Gets data as a byte stream over the specified IP Address and Port
number, and stores in the
        /// buffer.
        /// </summary>
        /// <returns></returns>
        public byte[] GetReceivedByteData()
        {
            client = listener.AcceptTcpClient();

            client.ReceiveBufferSize = privateReceiveBufferSize;
            //---get the incoming data through a network stream---
            nwStream = client.GetStream();
            buffer = new byte[client.ReceiveBufferSize];

            //---read incoming stream---
            bytesRead = nwStream.Read(buffer, 0, client.ReceiveBufferSize);

            return buffer;
        }

        #endregion

    }
}
```

Development Softwares Used

- <u>VISUAL STUDIO 2012</u>

  Visual Studio Software is a comprehensive software development IDE from Microsoft which allows for software development for any windows based platform in various languages.
  Since this application is .NET based, hence this IDE was utilised.

  This IDE is also the only IDE to support building, and live debugging of Windows Phone Apps on real devices.

CONCLUSION

The objective of this application was to provide a cheap and alternative solution to a wireless mic system usually deployed for public announcements.

Given the edge this application has over the the conventional system in terms of cost, range and versatility, it is safe to assume that the application is a feasible alternative and that it has successfully fulfilled it’s objective.

BIBLIOGRAPHY

- www.stackoverflow.com
- www.codeproject.com
- www.dreamincode.com
- www.msdn.microsoft.com
- www.google.com
- www.wikipedia.org

EXAMINER'S REMARKS