```
using RaspiHomeSpeechNSynthetize;
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
using System.Net;
using System.Net.Sockets;
using System.Text;
using System.Threading;
using System.Threading.Tasks;
using Windows.ApplicationModel.Core;
using Windows.Networking;
using Windows.Networking.Connectivity;
using Windows.Networking.Sockets;
using Windows.Storage.Streams;

namespace RaspiHomeSpeechNSynthetize
{
    public class CommunicationWithServer
    {
        #region Fields
        #region Constants
        // Default information to connect on the server
        private const int PORT = 54565;
        //// Need to be changed fo each configuration
        private const string IPSERVER = "10.134.97.117";// "192.168.2.8";

        private const string FORMATSTRING = "IPRasp={0};Location=
          {1};Component={2}";
        private const string COMMUNICATIONSEPARATOR = "@";

        // Important need to be changed if it's another room!
        private const string LOCATION = "Salon";
        private const string COMPONENT = "Microphone";
        private const string RPINAME = "Microphone_" + LOCATION;//
          "192.168.2.8";

        private const int MESSAGE_FULL_LENGHT = 512;
        #endregion

        #region Variables
        private Speecher _speecher;

        private StreamSocket _socket = new StreamSocket();
        private StreamSocketListener _listener = new StreamSocketListener();
        private List<StreamSocket> _connections = new List<StreamSocket>();
        private bool _isConnected = false;
        private bool _connecting = false;

        private string _messageCommand = "";
        #endregion
        #endregion

        #region Properties
        public Speecher Speecher
        {
```

```
55              get
56              {
57                  return _speecher;
58              }
59
60              set
61              {
62                  _speecher = value;
63              }
64          }
65
66          public StreamSocket Socket
67          {
68              get
69              {
70                  return _socket;
71              }
72
73              set
74              {
75                  _socket = value;
76              }
77          }
78
79          public StreamSocketListener Listener
80          {
81              get
82              {
83                  return _listener;
84              }
85
86              set
87              {
88                  _listener = value;
89              }
90          }
91
92          public List<StreamSocket> Connections
93          {
94              get
95              {
96                  return _connections;
97              }
98
99              set
100             {
101                 _connections = value;
102             }
103         }
104
105         public bool IsConnected
106         {
107             get
108             {
109                 return _isConnected;
110             }
```

```
111
112             set
113             {
114                 _isConnected = value;
115             }
116         }
117
118         public bool Connecting
119         {
120             get
121             {
122                 return _connecting;
123             }
124
125             set
126             {
127                 _connecting = value;
128             }
129         }
130
131         public string MessageCommand
132         {
133             get
134             {
135                 return _messageCommand;
136             }
137
138             set
139             {
140                 _messageCommand = value;
141             }
142         }
143         #endregion
144
145         #region Constructors
146         /// <summary>
147         /// Constructor: Initializer
148         /// </summary>
149         /// <param name="paramModel"></param>
150         public CommunicationWithServer(Speecher paramModel)
151         {
152             this.Speecher = paramModel;
153
154             Connect();
155         }
156         #endregion
157
158         #region Methods
159         /// <summary>
160         /// Connect the raspberry to the server
161         /// </summary>
162         private async void Connect()
163         {
164             try
165             {
166                 this.Connecting = true;
```

```
167                 await this.Socket.ConnectAsync(new HostName(IPSERVER),
                      PORT.ToString());
168                 SendForInitialize();
169                 this.Connecting = false;
170                 this.IsConnected = true;
171
172                 WaitForData(this.Socket);
173             }
174             catch (Exception)
175             {
176                 this.Connecting = false;
177                 this.IsConnected = false;
178             }
179         }
180
181         /// <summary>
182         /// Listen the traffic on the port
183         /// </summary>
184         private async void Listen()
185         {
186             this.Listener.ConnectionReceived += listenerConnectionReceived;
187             await this.Listener.BindServiceNameAsync(PORT.ToString());
188         }
189
190         void listenerConnectionReceived(StreamSocketListener sender,
              StreamSocketListenerConnectionReceivedEventArgs args)
191         {
192             this.Connections.Add(args.Socket);
193
194             WaitForData(args.Socket);
195         }
196
197         /// <summary>
198         /// Send the message in input to output
199         /// </summary>
200         /// <param name="socket"></param>
201         /// <param name="message"></param>
202         private async void SendMessage(StreamSocket socket, string message)
203         {
204             DataWriter dataWriter = new DataWriter(socket.OutputStream);
205             var len = dataWriter.MeasureString(message); // Gets the UTF-8
                  string length.
206             dataWriter.WriteInt32((int)len);
207             dataWriter.WriteString(message);
208             var ret = await dataWriter.StoreAsync();
209             dataWriter.DetachStream();
210         }
211
212         /// <summary>
213         /// Send to initialize the raspberry to the server
214         /// </summary>
215         private void SendForInitialize()
216         {
217             SendMessage(this.Socket, string.Format(COMMUNICATIONSEPARATOR +
                    RPINAME + COMMUNICATIONSEPARATOR + "Connection:" + FORMATSTRING,
                     GetHostName(), LOCATION, COMPONENT));
```

```
218         }
219
220         /// <summary>
221         /// Send the command to the server
222         /// </summary>
223         public void SendCommandToServer(string message)
224         {
225             SendMessage(this.Socket, COMMUNICATIONSEPARATOR + "Send:" +
                message);
226             this.MessageCommand = message;
227         }
228
229         /// <summary>
230         /// Wait data readed if exist
231         /// </summary>
232         /// <param name="socket"></param>
233         private async void WaitForData(StreamSocket socket)
234         {
235             DataReader dataReader = new DataReader(socket.InputStream);
236             dataReader.InputStreamOptions = InputStreamOptions.Partial;
237             var messageLenght = dataReader.UnconsumedBufferLength;
238             uint stringBytes = messageLenght;
239
240             try
241             {
242                 // Read modification in the stream
243                 stringBytes = await dataReader.LoadAsync(MESSAGE_FULL_LENGHT);
244
245                 // read message
246                 string messageRead = dataReader.ReadString(stringBytes);
247
248                 // Send in return if the value exist
249                 if (messageRead != "")
250                 {
251                     this.Speecher.ReplyForSynthetize(messageRead,
                        this.MessageCommand);
252                 }
253
254                 messageRead = "";
255             }
256             catch (Exception ) { }
257
258             WaitForData(socket);
259         }
260
261         /// <summary>
262         /// Get the ip of the raspberry
263         /// </summary>
264         /// <returns>return a string like 192.168.1.2</returns>
265         public string GetHostName()
266         {
267             List<string> IpAddress = new List<string>();
268             var Hosts =
                  Windows.Networking.Connectivity.NetworkInformation.GetHostNames
                  ().ToList();
269             foreach (var Host in Hosts)
```

```
270             {
271                 string IP = Host.DisplayName;
272                 IpAddress.Add(IP);
273             }
274             return IpAddress.Last();
275         }
276         #endregion
277     }
278 }
279
```