```
/*-------------------------------------------------*\
 * Author    : Salvi Cyril
 * Date      : 7th juny 2017
 * Diploma   : RaspiHome
 * Classroom : T.IS-E2B
 *
 * Description:
 *      RaspiHomePiFaceDigital2 is a program who use
 *   a PiFace Digital 2, it's an electronic card who
 *   can be use to plug electronic component. This
 *   program use the PiFace Digital 2 to activate
 *   light and store.
\*-------------------------------------------------*/

using System;
using System.Collections.Generic;
using System.Linq;
using Windows.Networking;
using Windows.Networking.Sockets;
using Windows.Storage.Streams;

namespace RaspiHomePiFaceDigital2
{
    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";

        // String format for connection of the client
        private const string FORMATSTRING = "Connection: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 RPINAME = "PiFace_" + LOCATION;

        private const int MESSAGE_FULL_LENGHT = 512;
        #endregion

        #region Variables
        private ModelPiFaceDigital2 _mPiFace;

        // Connection's variable
        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;
        #endregion
        #endregion

```

```
56          #region Properties
57          public ModelPiFaceDigital2 MPiFace
58          {
59              get
60              {
61                  return _mPiFace;
62              }
63
64              set
65              {
66                  _mPiFace = value;
67              }
68          }
69
70          public StreamSocket Socket
71          {
72              get
73              {
74                  return _socket;
75              }
76
77              set
78              {
79                  _socket = value;
80              }
81          }
82
83          public StreamSocketListener Listener
84          {
85              get
86              {
87                  return _listener;
88              }
89
90              set
91              {
92                  _listener = value;
93              }
94          }
95
96          public List<StreamSocket> Connections
97          {
98              get
99              {
100                 return _connections;
101             }
102
103             set
104             {
105                 _connections = value;
106             }
107         }
108
109         public bool IsConnected
110         {
111             get
```

```
112             {
113                 return _isConnected;
114             }
115 
116             set
117             {
118                 _isConnected = value;
119             }
120         }
121 
122         public bool Connecting
123         {
124             get
125             {
126                 return _connecting;
127             }
128 
129             set
130             {
131                 _connecting = value;
132             }
133         }
134         #endregion
135 
136         #region Constructors
137         /// <summary>
138         /// Constructor: Initializer
139         /// </summary>
140         /// <param name="paramModel"></param>
141         public CommunicationWithServer(ModelPiFaceDigital2 paramModel)
142         {
143             this.MPiFace = paramModel;
144 
145             Connect();
146         }
147         #endregion
148 
149         #region Methods
150         #region Methods
151         /// <summary>
152         /// Connect the raspberry to the server
153         /// </summary>
154         private async void Connect()
155         {
156             try
157             {
158                 this.Connecting = true;
159                 // wait a confirmation from the server
160                 await this.Socket.ConnectAsync(new HostName(IPSERVER),
                      PORT.ToString());
161                 SendForInitialize();
162                 this.Connecting = false;
163                 this.IsConnected = true;
164 
165                 WaitForData(this.Socket);
166             }
```

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

```
218         private async void WaitForData(StreamSocket socket)
219         {
220             DataReader dataReader = new DataReader(socket.InputStream);
221             dataReader.InputStreamOptions = InputStreamOptions.Partial;
222             var msglenght = dataReader.UnconsumedBufferLength;
223             uint stringBytes = msglenght;
224 
225 
226             try
227             {
228                 // Read modification in the stream
229                 stringBytes = await dataReader.LoadAsync(MESSAGE_FULL_LENGHT);
230 
231                 // read message
232                 string msg = dataReader.ReadString(stringBytes);
233 
234                 // Send in return if the value exist
235                 if (msg != "")
236                 {
237                     this.MPiFace.SetValue(msg);
238                 }
239             }
240             catch (Exception e)
241             {
242                 string output = e.Message;
243 
244                 if (msglenght < 1)
245                     return;
246             }
247 
248             // Restart loop to wait data
249             WaitForData(socket);
250         }
251 
252         /// <summary>
253         /// Get the ip of the raspberry
254         /// </summary>
255         /// <returns>return a string like 192.168.1.2</returns>
256         private string GetHostName()
257         {
258             List<string> IpAddress = new List<string>();
259             var Hosts =                                                    ↵
                Windows.Networking.Connectivity.NetworkInformation.GetHostNames  ↵
                ().ToList();
260             foreach (var Host in Hosts)
261             {
262                 string IP = Host.DisplayName;
263                 IpAddress.Add(IP);
264             }
265             return IpAddress.Last();
266         }
267 
268         /// <summary>
269         /// Get component in the list of components
270         /// </summary>
271         /// <returns> return a usable string for the connection on the      ↵
```

```
            server</returns>
272         private string GetComponent()
273         {
274             string result = "";
275             int cnt = 0;
276             foreach (var component in this.MPiFace.Components)
277             {
278                 // Get the name of the class
279                 result += component.ToString().Split('.').Last();
280                 cnt++;
281                 // Add the component separator for the string format
282                 if (cnt < this.MPiFace.Components.Count)
283                     result += ",";
284 
285             }
286 
287             return result;
288         }
289     }
290     #endregion
291     #endregion
292 }
293 
294 
```