

```
1 using System;
2 using System.Management;
3 using System.IO.Ports;
4 using System.Net;
5
6 namespace testProgram
7 {
8     class MainClass
9     {
10         //Local Network Settings Struct
11         public struct local_net
12         {
13             public string myMode, myHostname, myIPv4, mySubnet, myCIDR,
14                 myGateway;
15
16         static void Main()
17         {
18             local_net _net = new local_net();
19
20             while (true)
21             {
22                 get_info(ref _net);
23
24                 var portNames = SerialPort.GetPortNames();
25
26                 foreach (var port in portNames)
27                 {
28                     try
29                     {
30                         Console.Clear();
31                         Console.WriteLine("Configuration Dongle CONNECTED!");
32                         Console.WriteLine
33                             ("-----");
34                         Console.WriteLine("Communicating w/ " + port + "\n");
35
36                         //Setup COM Port
37                         SerialPort serialPort = new SerialPort(port, 9600,
38                             Parity.None, 8, StopBits.One);
39                         serialPort.DtrEnable = true;
40                         serialPort.Open();
41
42                         string rcv_data = "", rcv_ip = "", rcv_subnet = "";
43
44                         //Read Request from Dongle
45                         rcv_data = serialPort.ReadLine();
46
47                         //Parse Request
48                         if (rcv_data == "Mode\r")
49                         {
50                             serialPort.Write(_net.myMode);
51                             rcv_data = "";
52                         }
53                     }
54                     catch { }
55                 }
56             }
57         }
58     }
59 }
```

```
50     }
51
52     else if (rcv_data == "Host\r")
53     {
54         serialPort.Write(_net.myHostname);
55         rcv_data = "";
56     }
57
58     else if (rcv_data == "IP\r")
59     {
60         serialPort.Write(_net.myIPv4);
61         serialPort.DiscardOutBuffer();
62         serialPort.DiscardInBuffer();
63         rcv_data = "";
64     }
65
66     else if (rcv_data == "Sub\r")
67     {
68         serialPort.Write(_net.mySubnet);
69
70         serialPort.DiscardOutBuffer();
71         serialPort.DiscardInBuffer();
72         rcv_data = "";
73     }
74
75     else if (rcv_data == "Gate\r")
76     {
77         serialPort.Write(_net.myGateway);
78
79         serialPort.DiscardOutBuffer();
80         serialPort.DiscardInBuffer();
81         rcv_data = "";
82     }
83
84     else if (rcv_data == "DHCP\r")
85     {
86         serialPort.Write("OK");
87         setDHCP();
88         serialPort.Write("OK");
89
90         serialPort.DiscardOutBuffer();
91         serialPort.DiscardInBuffer();
92     }
93
94     else if (rcv_data == "Static\r")
95     {
96         serialPort.Write("OK");
97         rcv_ip = serialPort.ReadLine();
98         rcv_ip = rcv_ip.Replace("\r\n", "").Replace("\r",
99         "").Replace("\n", "");
100
101         serialPort.Write("OK");
```

```

101         rcv_subnet = serialPort.ReadLine();
102         rcv_subnet = rcv_subnet.Replace("\r\n", "").Replace  ➤
        ("\r", "").Replace("\n", "");
103
104         setStatic(rcv_ip, rcv_subnet);
105         serialPort.Write("OK");
106
107         serialPort.DiscardOutBuffer();
108         serialPort.DiscardInBuffer();
109     }
110
111     //Debugging purposes prints to console so I can see the  ➤
    info
112     Console.WriteLine("Hostname: " + _net.myHostname);
113     Console.WriteLine("IP: " + _net.myIPv4 + "/" +  ➤
    _net.myCIDR);
114     Console.WriteLine("Subnet: " + _net.mySubnet);
115     Console.WriteLine("Gateway: " + _net.myGateway + "\n");
116
117     //close Port
118     serialPort.Close();
119     break;
120 }
121 catch (Exception ex)
122 {
123     Console.Clear();
124     Console.WriteLine("Configuration Dongle DISCONNECTED!");
125     Console.WriteLine  ➤
    ("-----");
126     Console.WriteLine("Error opening port " + port + ": {0}",  ➤
    ex.Message);
127 }
128 }
129 }
130 }
131
132 public static void get_info(ref local_net _net)
133 {
134     //Retrieve Hostname
135     IPHostEntry hostInfo = Dns.GetHostEntry(Dns.GetHostName());
136     _net.myHostname = hostInfo.HostName;
137
138     //Retrieve IPv4 Address of Hostname
139     IPAddress[] address = Dns.GetHostAddresses(Dns.GetHostName());
140
141     //Set network settings struct values to NULL
142     _net.myMode = "No Internet Access";
143     _net.myIPv4 = address[1].ToString(); //Default IPv4 address 127.0.0.1  ➤
    w/ no network connection
144     _net.mySubnet = " ";
145     _net.myCIDR = " ";
146     _net.myGateway = " ";

```

```
147
148     //Creating instance of ManagementClass for network adapter settings
149     ManagementClass objMC = new ManagementClass
150         ("Win32_NetworkAdapterConfiguration");
151
152     //Gets all the info for all the network adapters
153     ManagementObjectCollection objMOC = objMC.GetInstances();
154
155     //Parse through the info to find the network adapter with Network
156     connection
157     foreach (ManagementObject objMO in objMOC)
158     {
159         //If an IP exists in one of the adapters, then that's the active
160         network we are working with
161         if ((bool)objMO["IPEnabled"])
162         {
163             try
164             {
165                 //Get Mode, IPv4 Address, Subnet Mask, Gateway
166                 string mode = ((bool)objMO["DHCPEnabled"]).ToString
167                     ().ToLower() == "true" ? "DHCP" : "Static";
168                 string[] ipaddress = (string[])objMO["IPAddress"];
169                 string[] subnet = (string[])objMO["IPSubnet"];
170                 string[] gateway = (string[])objMO["DefaultIPGateway"];
171
172                 //Assign struct values w/ the values retrieved from code
173                 above
174                 _net.myMode = mode;
175                 _net.myIPv4 = ipaddress[0];
176                 _net.mySubnet = subnet[0];
177
178                 //Assign Gateway w/ a try function since gateway is
179                 optional during STATIC mode
180                 try
181                 {
182                     if (gateway == null)
183                         _net.myGateway = "unavailable";
184                     else
185                         _net.myGateway = gateway[0];
186                 }
187                 catch (Exception)
188                 {
189                     _net.myGateway = "unavailable";
190                     throw;
191                 }
192
193                 //Calculate CIDR from Subnet
194                 string[] tokens = _net.mySubnet.Split('.');
195                 string result = "";
196                 foreach (string token in tokens)
197                 {
198                     int tokenNum = int.Parse(token);
199                     string octet = Convert.ToString(tokenNum, 2);
```

```

193         while (octet.Length < 8)
194             octet = octet + '0';
195         result += octet;
196     }
197
198     //Assign CIDR struct
199     _net.myCIDR = (result.LastIndexOf('1') + 1).ToString();
200
201     }
202     catch (Exception)
203     {
204         throw;
205     }
206 }
207
208 }
209
210 public static void setDHCP()
211 {
212     //Creating instance of ManagementClass for network adapter settings
213     ManagementClass objMC = new ManagementClass("Win32_NetworkAdapterConfiguration");
214     //Gets all the info for all the network adapters
215     ManagementObjectCollection objMOC = objMC.GetInstances();
216
217     //Parse through the info to find the network adapter with Network
218     //connection
219     //If an IP exists in one of the adapters, then that's the active
220     //network we are working with
221     foreach (ManagementObject objMO in objMOC)
222     {
223         if ((bool)objMO["IPEnabled"])
224         {
225             try
226             {
227                 //Enable DHCP
228                 var ndns = objMO.GetMethodParameters("SetDNSServerSearchOrder");
229                 ndns["DNSServerSearchOrder"] = null;
230                 objMO.InvokeMethod("EnableDHCP", null, null);
231                 objMO.InvokeMethod("SetDNSServerSearchOrder", ndns, null);
232             }
233             catch (Exception)
234             {
235                 throw;
236             }
237         }
238     }
239
240     public static void setStatic(string ip_address, string subnet_mask)

```

```

240     {
241         //Creating instance of ManagementClass for network adapter settings
242         ManagementClass objMC = new ManagementClass           ↗
243             ("Win32_NetworkAdapterConfiguration");
244         //Gets all the info for all the network adapters
245         ManagementObjectCollection objMOC = objMC.GetInstances();
246
247         //Parse through the info to find the network adapter with Network ↗
248         //connection
249         //If an IP exists in one of the adapters, then that's the active ↗
250         //network we are working with
251         foreach (ManagementObject objMO in objMOC)
252         {
253             if ((bool)objMO["IPEnabled"])
254             {
255                 try
256                 {
257                     ManagementBaseObject setIP;
258                     ManagementBaseObject newIP = objMO.GetMethodParameters           ↗
259                         ("EnableStatic");
260
261                     //Set IPv4 Address and Netmask recieved from ↗
262                     Configuration Dongle
263                     newIP["IPAddress"] = new string[] { ip_address };
264                     newIP["SubnetMask"] = new string[] { subnet_mask };
265
266                     //Enable Static Mode
267                     setIP = objMO.InvokeMethod("EnableStatic", newIP, null);
268                 }
269                 catch (Exception)
270                 {
271                     throw;
272                 }
273             }
274         }
275     }
276
277     public static UInt16 ModRTU_CRC(string buf, int len)
278     {
279         UInt16 crc = 0xFFFF;
280
281         for (int pos = 0; pos < len; pos++)
282         {
283             crc ^= (UInt16)buf[pos];           // XOR byte into least sig. ↗
284             byte of crc
285
286             for (int i = 8; i != 0; i--)
287             {
288                 // Loop over each bit
289                 if ((crc & 0x0001) != 0)
290                 {
291                     // If the LSB is set
292                     crc >>= 1;           // Shift right and XOR ↗
293                     0xA001
294                 }
295             }
296         }
297     }

```

```
285         crc ^= 0xA001;
286     }
287     else                                     // Else LSB is not set
288         crc >>= 1;                         // Just shift right
289     }
290 }
291 // Note, this number has low and high bytes swapped, so use it
292 // accordingly (or swap bytes)
293 return crc;
294 }
295 }
296
297
```