PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE
ESCUELA DE INGENIERÍA
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN
MECOLAB

Tutorial: Conexión Gadgeteer – Xamarin Android

Vicente Opaso Valenzuela
vaopaso@uc.cl
Octubre de 2015

Índice

Introducción

Para comenzar con este tutorial, primero contextualizaremos e introduciremos al lector algunas definiciones importantes a saber para desarrollar este proyecto.

El mundo Gadgeteer consiste en una plataforma que mediante un microcontrolador se conectan distintos sensores o actuadores que se pueden controlar mediante el programa cargado a esta placa o microcontrolador. Este microcontrolador funciona con .NET Microframework. Para más información: https://www.ghielectronics.com/technologies/gadgeteer

Por otro lado, Xamarin es una plataforma de desarrollo que nos permite desarrollar en C# aplicaciones móviles para Android, iOS o WindowsPhone.

En este tutorial aprenderemos a desarrollar un programa en .NET Gadgeteer y una aplicación en Xamarin para Android, que pueda controlar inalámbricamente un microcontrolador Gadgeteer para dar determinadas respuestas como prender una luz o leer la información de un sensor. Para ello se utilizará un módulo WiFi conectado a la placa que haga la conexión entre el dispositivo Android y la placa Gadgeteer. En lo que respecta a la conexión WiFi, el tutorial solo está adaptado para funcionar dentro de una red local, es decir, el servidor que estará alojado en la placa y el dispositivo Android, estarán conectados a la misma red.

El desarrollo del tutorial será con programas del sistema operativo Windows. Además, el código estará basado en el lenguaje C# tanto para Gadgeteer como para Xamarin.

No nos preocuparemos mucho por el diseño y manejo de la aplicación Android, sino que el tutorial se enfocará en explicar en detalle el código de la conexión *Mobile-Gadgeteer* y su funcionamiento por vía WiFi.

Materiales necesarios:

- FEZ Spider mainboard
- Módulo WiFi RS21
- Módulo USB Client SP (o DP)
- Módulo LightSense
- Módulo DistanceUS3
- Módulo Relay X1

Todos estos módulos son de GHI Electronics, aunque también existen otros de empresas como Seeed Studio u otros.
También necesitaremos los cables USB correspondientes de la placa y del dispositivo móvil, para poder conectarlos al computador.

Cuando tengamos los materiales necesarios, podemos proceder al desarrollo.

Paso 1: Instalaciones

1.1) MicroFramework .NET Gadgeteer

Primero que nada, hay que instalar en el computador los programas para trabajar: Visual Studio y el soporte para .NET Gadgeteer, que nos permitirán programar la placa.
Seguir cuidadosamente las instrucciones en este link:
https://www.ghielectronics.com/support/netmf

1.2) Xamarin Android

Utilizaremos Xamarin Android para poder desarrollar la aplicación móvil que utilizaremos. Para ello, en este tutorial se utilizará el programa Xamarin Studio para Windows.

Descargamos el programa directamente desde la página oficial de Xamarin:
http://xamarin.com/download
Probablemente te pedirá hacer una cuenta. Una vez descargado, se siguen todas las instrucciones de instalación que ahí se indican.

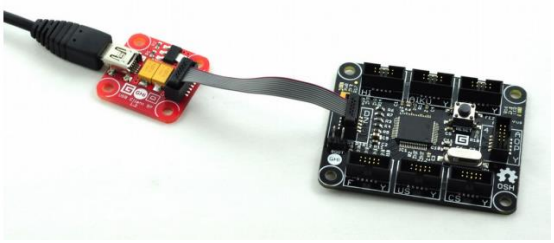
También existe la posibilidad de trabajar Xamarin Android con Visual Studio, pero para ello hay que instalarle una extensión, y además pueden haber problemas al trabajar y debuggear con Visual Studio dependiendo del tipo de cuenta que tengas en Xamarin (con una cuenta básica no te deja hacer Debug). Para más información: http://xamarin.com/visual-studio

1.3) Modo de conexión

Primero, hay que proceder a conectar la placa al computador. Para esto hay que saber conectar físicamente los componentes Gadgeteer entre sí.

No se puede conectar cualquier módulo a la placa cuando uno quiera y como uno quiera: De partida hay que asegurarse que la placa esté desconectada de cualquier fuente de alimentación. Luego observamos al costado del pin de cada módulo que sale una letra (o más de una). Además en la placa también salen letras al costado de cada pin de conexión. Entonces intuitivamente se conecta el módulo solamente a los pines de la placa que contengan su letra, es decir, si tuviéramos un módulo con letras X e Y, entonces buscamos en la placa algún pin que contenga alguna de estas letras y lo conectamos (no es necesario que contenga las dos letras, sólo una es suficiente).

Es importante notar que hay módulos que son de color rojo. Estos representan a los módulos de energía que alimentan a la placa. Hay que asegurarse que la placa tenga sólo un módulo rojo conectado.

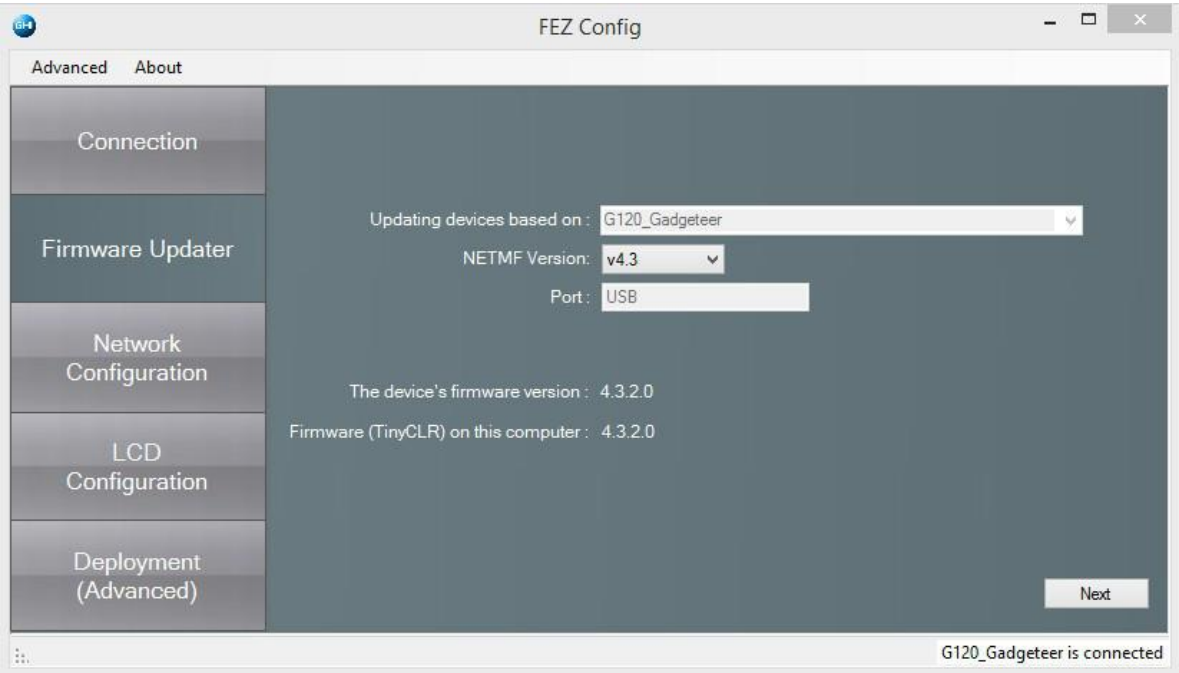


Una vez aprendido el modo de conexiones, conectamos el módulo USB Client SP con la FEZ Spider, y luego lo conectamos mediante USB al computador.

1.4) Actualizar Firmware

Nos vamos a la carpeta donde se instaló *GHI Electronics* (usualmente en *Archivos de Programa*), luego abrimos la carpeta *GHI FEZ Config*. Aquí ejecutamos el programa FEZ Config.exe

Nos vamos a la pestaña de Firmware Updater



Si tenemos conectado el dispositivo, podemos observar la versión actual que tiene la placa. En este caso vamos a trabajar con la versión 4.3 (la más actual hasta la fecha de este tutorial), entonces si el firmware actual de la placa es más antiguo, hay que actualizar la placa seleccionando *NETMF Version: v4.3* y pulsar *Next* y seguir cuidadosamente las instrucciones para no dañar la placa. Si el firmware ya es de la versión que queremos, entonces omitir este último paso.

Una vez con el firmware deseado, podremos empezar a trabajar.
Como a continuación iremos a desarrollar el programa, no es necesario que siga conectada la placa al computador, por lo que luego de completar este proceso de actualización de firmware, podemos desconectar la placa del puerto USB.

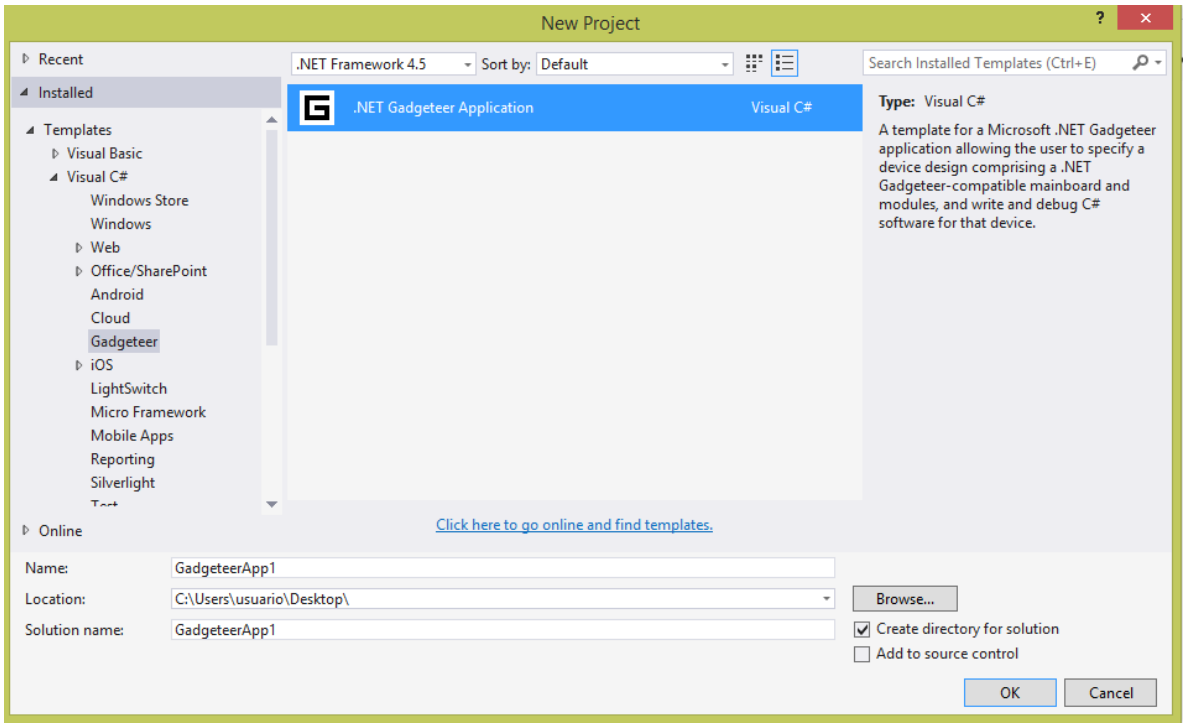
Errores comunes:

- En la sección 1.1 si no funciona el programa, será mejor desinstalar todo y volver a realizarlo siguiendo cuidadosamente los pasos, en orden y respetando las versiones indicadas.
- En el modo de conexiones en la sección 1.3, si al conectar la placa al computador ésta se desconecta repentinamente y luego se vuelve a conectar, y así sucesivamente en alguna especie de sin control, entonces puede ser la alimentación de corriente de la placa. Esto suele suceder cuando tenemos muchos módulos conectados y no solo uno como ahora. Para solucionar esto, hay que cambiar el módulo USB Client SP por el USB Client DP y alimentar la placa con 12 volts (ver especificaciones en: https://www.ghielectronics.com/catalog/product/280), mediante algún transformador conectado a la pared y al módulo USB Client DP.

Paso 2: Iniciar proyecto Gadgeteer

2.1) Seleccionar Template

Abrir Visual Studio 2013 (o la versión adecuada). Vamos a *New Project*. Luego nos aparecerá la siguiente ventana:

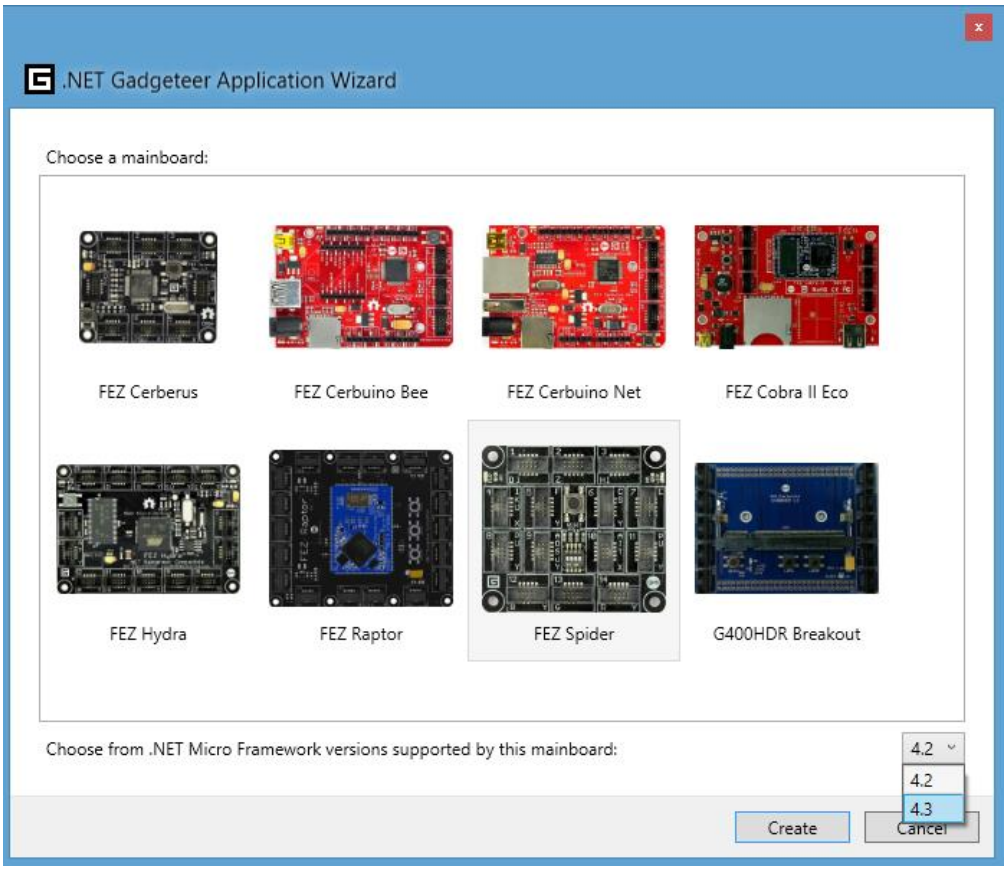


En la barra de navegación a mano izquierda, en *Templates* seleccionamos *Gadgeteer*. Luego seleccionamos en la ventana principal *.NET Gadgeteer Application* (la única opción que aparece en la imagen). Luego elegimos el nombre de la aplicación, su locación, y pulsamos *OK*.

2.2) Seleccionar placa y versión

Ahora nos aparecerá otra ventana.

Atención: Aquí hay que elegir la placa que programaremos, y su versión. Primero hay que elegir la versión que tiene la placa en la esquina inferior derecha. Esto es muy importante, porque a la placa hay que programarla en la misma versión de su firmware instalado. En el paso anterior establecimos que se trabajará con la versión 4.3, entonces seleccionamos esta versión dentro de las opciones.



Luego seleccionamos FEZ Spider, que es la placa con que trabajaremos y pulsamos *Create*.

Errores comunes:

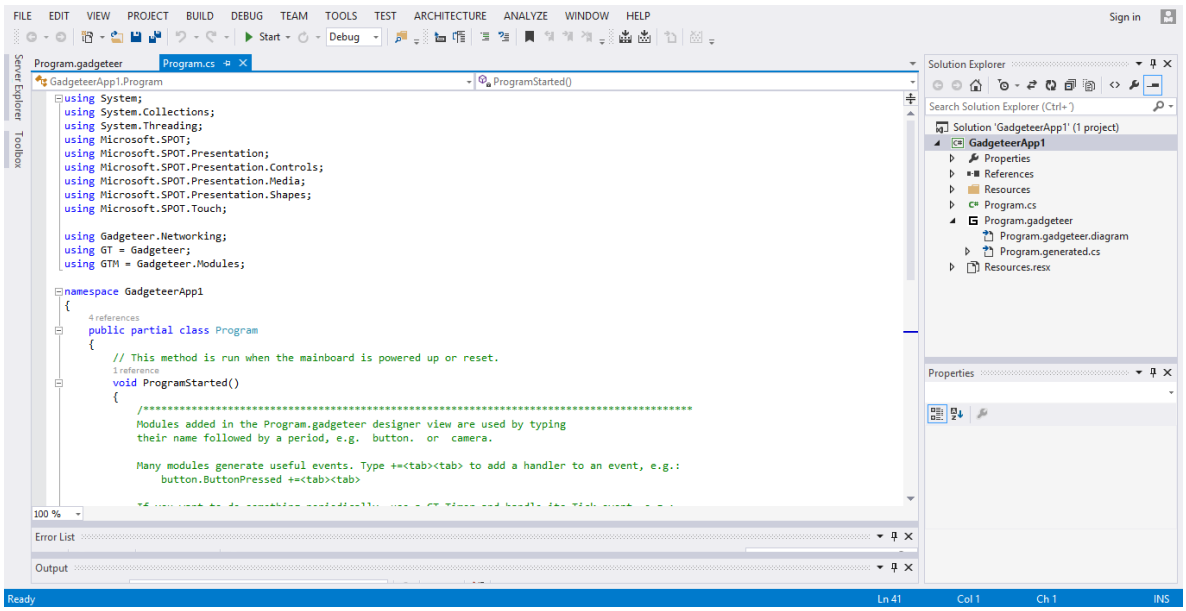
- En la ventana de nuevo proyecto en la sección 2.1, si la opción *Gadgeteer* no aparece en *Templates*, entonces revisar errores comunes de la sección 1.1 (problemas de instalación).

Paso 3: Programa Gadgeteer WiFi

3.1) Empezar las conexiones

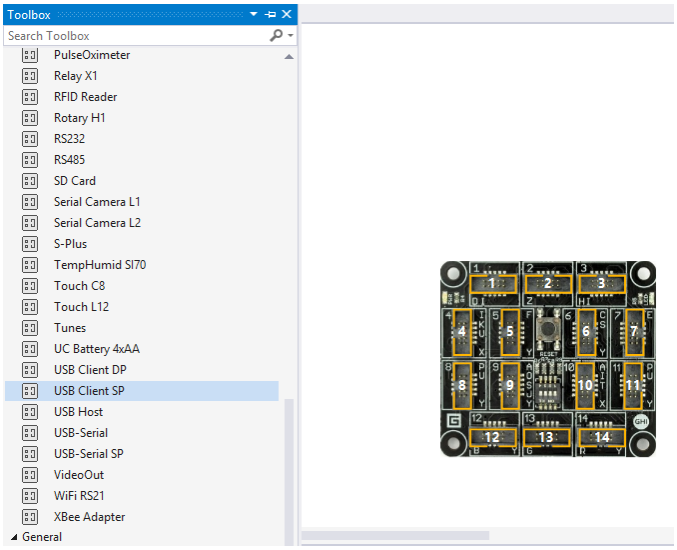
Ahora empezaremos a hacer el programa que permitirá obtener información de los sensores de luz y distancia, además del control de un relé, todos estos conectados a la placa.

Cuando iniciamos un proyecto Gadgeteer, nos aparecerá una imagen como la siguiente:



Como se observa, existen dos pestañas. La actual (*Program.cs*) es la clase principal con que se ejecuta el programa en el microcontrolador, específicamente lo que se ejecuta al iniciarse es el método *void ProgramStarted().*

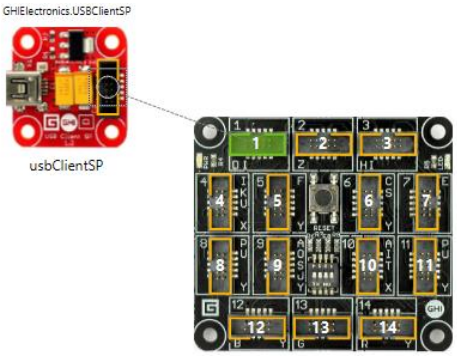
Por otro lado tenemos la pestaña *Program.gadgeteer*, que es donde se harán las conexiones de los módulos dentro del programa. Entonces ahora nos vamos a esta pestaña.



Se observa la imagen de la placa elegida, que en nuestro caso es la FEZ Spider, y a un costado se encuentra una barra de herramientas que contiene los componentes necesarios para poder trabajar.

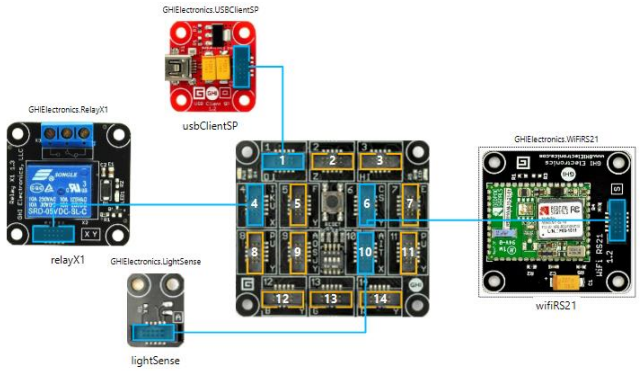
Entonces dentro de la barra de herramientas, en la sección de *GHI Electronics*, buscamos el módulo *USB Client SP*.

Luego lo arrastramos hacia la hoja de trabajo al lado de la placa. Entonces para efectuar la conexión simplemente seleccionamos el pin del módulo y arrastramos el "conector" que aparece, hacia algún pin compatible de la placa. En este caso, como el módulo es de energía y su letra es D, entonces el único pin compatible será el 1 en la placa (los pines posibles se activan de color verde cuando se hace la conexión).



Una vez conectado el módulo de alimentación, hacemos lo mismo con los otros módulos a utilizar. Primero buscamos el sensor de luz *LightSense* en la barra de herramientas, luego se desplaza hacia la hoja de trabajo y se realiza la conexión. Como el *LightSense* es de letra A, es posible conectarlo a los pines 9 o 10 de la placa. Para el caso de este tutorial, lo conectaremos en el pin 10. Lo mismo hacemos con el módulo *WiFi RS21* que lo conectaremos al pin 6, y con el módulo *Relay X1* que lo conectaremos al pin 4.

Atención: Claramente, hay que ser consistente entre la conexión que se está realizando en el programa y la que se conectará físicamente, chequeando siempre que los pines estén bien conectados (ver sección 1.3). Aunque como ahora sólo estamos programando (sin la placa conectada al computador), la conexión física puede realizarse más adelante.



Una vez realizadas estas cuatro conexiones, deberíamos tener algo como la imagen del costado.

Entonces ahora se prosigue a conectar el sensor de distancia *DistanceUS3*, pero sucede que tenemos un problema: por más que lo buscamos en la barra de herramientas, este no se encuentra dentro de las opciones. Entonces a continuación explicaremos como solucionar esto.

Nota: esto puede suceder con varios módulos, sobre todo con los que no son de la empresa GHI Electronics. La solución entregada en la siguiente sección es general para todo tipo de módulos.

3.2) Conexión y uso de driver

Para utilizar un módulo que no se encuentra en las opciones en la pestaña de conexiones *Program.gadgeteer*, entonces tendremos que descargar el driver correspondiente en internet para trabajar con ese sensor. En la siguiente dirección encontrarás más información y un repositorio para descargar algunos drivers:
https://www.ghielectronics.com/docs/122/gadgeteer-driver-modification
En nuestro caso, además del *LightSense*, trabajaremos con el sensor de distancia *DistanceUS3*, que no se encuentra dentro de las opciones en el diagrama de conexiones. En la solución de ejemplo adjunta al final del tutorial, se encuentra el driver incluido (*Distance_US3.cs*). Al incorporar este driver a la solución, podemos acceder a todos los métodos y funciones necesarias para poder trabajar con el sensor.

Como este módulo viene por driver externo, la conexión no se hace por el diagrama de conexiones como se hacía con los otros módulos ya conectados, sino que se hace en el mismo código del programa. Luego, como el sensor de distancia es compatible con el pin 8 de la placa (letra Y), se procederá a conectar ahí. Entonces en las variables de la clase se define:

```
public partial class Program
{
    Distance_US3 distanceUS3 = new Distance_US3(8);

    void ProgramStarted()
    {...}
    ...
}
```

Pasándole como parámetro a la clase *Distance_US3* (que tiene que ser pública) el número del pin donde estamos conectando físicamente este módulo. Luego se crea el objeto *distanceUS3* de clase *Distance_US3* (driver) que contiene las funciones y lo necesario para trabajar con el sensor de distancia.

Atención: En este procedimiento de conexión, hay que asegurarse que en el diagrama de conexiones no exista nada conectado en el pin a utilizar (en este caso el 8), sino que solamente se conecta desde el código del programa como se acaba de indicar.

3.3) Servidor WiFi

Una vez hecha las conexiones en el software, procedemos a construir el servidor que estará alojado en la placa.
Primero definimos las variables de la clase que iremos a utilizar y completamos el método de inicio del programa:

```
public partial class Program
{
    private string IPAddress;
    private bool serverStarted = false;
    private string SSID = "Vicente";
    private string PASSWORD = "12345678";
    Distance_US3 distanceUS3 = new Distance_US3(8);

    void ProgramStarted()
    {
        Debug.Print("Program Started");

        if (!serverStarted)
            StartServer(); //Begin the connection
        else
            StopServer(); //Stops the connection
    }
}
```

IPAddress contendrá el IP que se le asignará al módulo WiFi, es decir, el IP del servidor que se creará en la placa. Luego *serverStarted* indica que por defecto el servidor no está iniciado cuando parte el programa. Además sabemos que las redes WiFi tienen un identificador o SSID (nombre de la red) y una contraseña, por lo que en los valores de *SSID* y *PASSWORD* hay que colocar los datos

correspondientes de la red que nos queramos conectar. A modo de ejemplo, en este tutorial nos conectaremos a la red *"Vicente"* que tiene como contraseña *"12345678"*.

Luego en *ProgramStarted()*, se hacen llamados a métodos que todavía no están definidos, entonces ahora procedemos a definir los métodos de partida y de detención del servidor. Agregamos a la clase el método *StartServer()* y *StopServer()*:

```
using Microsoft.SPOT.Net.NetworkInformation;
...

private void StartServer()
{
    if (!wifiRS21.NetworkInterface.Opened)
    { wifiRS21.NetworkInterface.Open(); }

    if (!wifiRS21.NetworkInterface.IsDhcpEnabled)
    { wifiRS21.NetworkInterface.EnableDhcp(); }

    if (!wifiRS21.NetworkInterface.LinkConnected) //If there is still no established connection
    {
        //Assigning events that handle connection
        NetworkChange.NetworkAddressChanged += NetworkChange_NetworkAddressChanged;
        NetworkChange.NetworkAvailabilityChanged += NetworkChange_NetworkAvailabilityChanged;

        wifiRS21.UseThisNetworkInterface();
        wifiRS21.NetworkInterface.Join(SSID, PASSWORD); //Joining the desired network

        Debug.Print("Connecting to network...");
        while (wifiRS21.NetworkInterface.IPAddress == "0.0.0.0")
        { Thread.Sleep(250); } //The while remains running until is assigned an IP
        Debug.Print("Connected!. SSID: " + SSID);
    }
    else //If there is already an established connection
    { RunServer(); }
}

private void StopServer()
{
    WebServer.StopLocalServer();
    serverStarted = false;
    Debug.Print("Server stopped");
}
```

La primera vez que se inicia el programa, se ejecuta el método *StartServer()*. Aquí se hacen las configuraciones necesarias para poder establecer una conexión y luego como todavía no existe una establecida, ingresa a la tercera condición *(if LinkConnected)* y asigna los eventos *NetworkAddressChanged* y *NetworkAvailabilityChanged* que tienen que contener lo siguiente:

```
void NetworkChange_NetworkAddressChanged(object sender, EventArgs e)
{
    Debug.Print("Network address changed");
    IPAddress = wifiRS21.NetworkInterface.IPAddress;
    Debug.Print("Network address changed. IP: " + IPAddress);
    RunServer();
}
```

```
void NetworkChange_NetworkAvailabilityChanged(object sender, NetworkAvailabilityEventArgs e)
{
    Debug.Print("Network availability: " + e.IsAvailable.ToString());
}
```

En el primer evento *NetworkAddressChanged* se observa que cuando se asigna una IP, se ejecuta luego el método *RunServer()*, pero este todavía no está definido, así que luego nos preocuparemos de este método.

Luego de asignar los eventos en *StartServer()*, el módulo WiFi con el método *Join()* trata de unirse a la red de nombre contenido en la variable *SSID* y con contraseña en la variable *PASSWORD*.

Atención: Si el módulo WiFi no encuentra ninguna red con esas características, ya sea porque hay un dato mal escrito o porque la red no se encuentra dentro del rango de señal, entonces el programa lanzará una excepción y se caerá. Por lo tanto hay que asegurarse que la red esté dentro del rango, que esté disponible para conectarse y definir los datos sin errores dentro de las variables del código. Para evitar que el programa se caiga, se podría tratar de poner esta línea de código en un *try – catch*, pero de todos modos, si no se conecta habría que reiniciar el programa o replantear el código con un *Timer* para que trate de conectarse constantemente.

Luego de captar la red, el programa espera hasta que se asigne correctamente una IP estable, y en ese caso se activa el evento *NetworkAddressChanged* guardando en nuestra variable *IPAddress* la IP asignada y luego ejecutando el método *RunServer()*.

Entonces ahora tenemos que definir el método que hará correr al servidor:

```
private void RunServer()
{
    Debug.Print("Starting Server");
    WebEvent GetValueEventSensors = WebServer.SetupWebEvent("Sensors");
    WebEvent GetValueEventRelay = WebServer.SetupWebEvent("Relay");

    GetValueEventSensors.WebEventReceived += GetValueEventSensors_WebEventReceived;
    GetValueEventRelay.WebEventReceived += GetValueEventRelay_WebEventReceived;

    WebServer.StartLocalServer(IPAddress, 80);
    serverStarted = true;
}
```

Aquí se ejecuta *WebServer.SetupWebEvent(path)* que crea una dirección en el servidor, para que se puedan procesar peticiones hechas a este. La dirección será http://{IP}:{port}/{path} donde *{IP}* es la IP previamente asignada, *{port}* es un puerto a definir y *{path}* es el nombre de la ruta asignada, en este caso "*Sensors*" o "*Relay*" dependiendo de la petición.

Luego se definen los eventos *WebEventReceived* que recibirán una petición (luego los definiremos), y finalmente se inicia el servidor en la IP asignada, en el puerto 80 para ejemplo de este tutorial. Entonces una vez comenzado el servidor, cualquier dispositivo conectado a la misma red, puede hacerle una petición GET al servidor a la dirección establecida. Para dar un ejemplo, supongamos que la IP asignada es - 192.168.0.3 - entonces si quisiéramos hacer alguna petición desde otro dispositivo para leer los sensores, tendríamos que hacerla a http://192.168.0.3:80/Sensors

Cuando llegue una petición al servidor se ejecutará el evento *WebEventReceived*. En este caso como queremos controlar el relé o leer los sensores, definiremos dos eventos, los que tienen que contener lo siguiente:

```
void GetValueEventSensors_WebEventReceived(string path, WebServer.HttpMethod method, Res-
ponder responder)
{
    Debug.Print("Received request");
    string distance = distanceUS3.GetDistanceInCentimeters(3).ToString();
    string light = (lightSense.ReadProportion()*100).ToString();
    if (light.Length > 4) { light = light.Substring(0, 4); } //only to round off

    responder.Respond(light+"/"+distance); //Responding (sensor value)
    Debug.Print("Light: " + light + "%. Dist: "+distance+" cm.");
    Debug.Print("Sending response");
}

void GetValueEventRelay_WebEventReceived(string path, WebServer.HttpMethod method, Respon-
der responder)
{
    Debug.Print("Received request");
    if (relayX1.Enabled)
    { relayX1.TurnOff(); }
    else
    { relayX1.TurnOn(); }
}
```

El evento destinado a los sensores, primero hace las mediciones de los componentes conectados: en este caso toma la distancia en centímetros que mide el *DistanceUS3* pasándole como parámetro el número 3 (puede cambiar) que es el número de mediciones que queremos que haga para luego hacer un promedio. Luego obtiene la proporción de luz que está midiendo el *LightSense*.

Nota: En el driver de distancia utilizado existe este método de medir en centímetros, pero tal vez en otro driver no esté este método, pero se pueden utilizar otros. En general, dependiendo del tipo de driver o módulo existen algunos que poseen varios tipos de medición o características, por ejemplo, el *LightSense* utilizado puede medir la luz medida en una escala de *lux* y también puede medir la proporción de luz en porcentaje (utilizamos esta última porque es más intuitiva).

Luego con el método *Respond()* se procede a responderle al dispositivo con la información obtenida de los sensores.

Atención: Hemos dado una respuesta en formato *string* y de la forma "*{luz}/{distancia}*". Esto hay que tenerlo presente, ya que al dispositivo móvil que haga la petición le llegará tal cual este *string*, por lo que tenemos que saber interpretarlo. Lo veremos más adelante durante el desarrollo de la aplicación móvil.

Además, en el evento destinado al control del relé, observamos que el código primero evalúa si el relé está prendido o no. En caso afirmativo, entonces lo apaga, de otro modo lo enciende.

Nota: Cuando se refiere a apagar o prender el relé, en realidad quiere decir que si deja pasar la corriente o no. Si tuviéramos cableado el relé a una ampolleta, entonces sería apagar o prender la ampolleta. De todos modos se puede probar este programa sin conectar el relé a nada, porque el módulo *RelayX1* trae consigo un led que indica si está prendido o apagado.

3.4) Servidor UDP

Ya hemos terminado casi todo lo correspondiente a la programación en Gadgeteer. Lo único que nos falta es hacer un servidor UDP para que la aplicación móvil pueda comunicarse con la placa, ya que a priori este no sabe la IP que tiene la placa, por lo tanto no podrá mandarle peticiones sin saber su dirección.

Para solucionar esto, cuando la aplicación móvil se encienda, ésta hará un UDP broadcast mandando una señal (pidiendo la IP) hacia todas las IP de la red local, en un puerto específico. De este modo, la placa Gadgeteer estará escuchando en todo momento si recibe alguna señal que le pida su IP, y en ese caso, ésta responderá con su dirección IP, de modo que la aplicación móvil pueda recibir la respuesta de la placa, y guardar su IP para hacer uso de ella en el programa.

Entonces tenemos que hacer un servidor UDP en la placa, que escuche en todo momento señales entrantes y que además sea capaz de responderlas en caso necesario.

Creamos la clase *UdpSocketServer* y *DataReceivedEventArgs*:

```
using System;
using System.Net.Sockets;
using System.Net;
…
public class UdpSocketServer
{
    public const int DEFAULT_SERVER_PORT = 5123;
    private int port;
    private Socket socket;

    public delegate void DataReceivedEventHandler(object sender, DataReceivedEventArgs e);
    public event DataReceivedEventHandler DataReceived;

    public UdpSocketServer()
        : this(DEFAULT_SERVER_PORT)
    { }

    public UdpSocketServer(int port)
    {
        this.port = port;
        socket = new Socket(AddressFamily.InterNetwork, SocketType.Dgram, ProtocolType.Udp);
    }

    private void OnDataReceived(DataReceivedEventArgs e)
    {
        if (DataReceived != null)
            DataReceived(this, e);
    }
}
```

```
public class DataReceivedEventArgs : EventArgs
{
    public EndPoint RemoteEndPoint { get; private set; }
    public byte[] Data { get; private set; }
    public byte[] ResponseData { get; set; }

    public DataReceivedEventArgs(EndPoint remoteEndPoint, byte[] data)
    {
        RemoteEndPoint = remoteEndPoint;
        if (data != null)
        {
            Data = new byte[data.Length];
            data.CopyTo(Data, 0);
        }
    }
}
```

Como se observa en la primera clase, por defecto el servidor se inicializa en el puerto 5123 (la utilizaremos como ejemplo en este tutorial) y se asigna el evento necesario. La segunda clase es para manejar la señal recibida por el servidor, en que luego ésta ejecutará una respuesta de ser necesario. Ahora nos falta agregar métodos para la clase *UdpSocketServer* que se encargarán de iniciar el servidor, ejecutar respuesta, y el uso interno de la clase. De este modo, dentro de la clase *UdpSocketServer* ponemos los siguientes métodos:

```
public void Start()
{
    new Thread(StartServerInternal).Start();
}

private void StartServerInternal()
{
    EndPoint endPoint = new IPEndPoint(IPAddress.Any, port);
    socket.Bind(endPoint);

    while (true)
    {
        if (socket.Poll(-1, SelectMode.SelectRead))
        {
            byte[] buffer = new byte[socket.Available];
            int bytesRead = socket.ReceiveFrom(buffer, ref endPoint);
            string ss = endPoint.ToString();
            IPAddress ip = IPAddress.Parse(ss.Split(':')[0]);
            endPoint = new IPEndPoint(ip, port);
            DataReceivedEventArgs args = new DataReceivedEventArgs(endPoint, buffer);
            OnDataReceived(args);

            if (args.ResponseData != null)
                Thread.Sleep(500);
                socket.SendTo(args.ResponseData, endPoint);
        }
        else
        {
            Thread.Sleep(10);
        }
    }
}
```

```
public static string BytesToString(byte[] bytes)
{
    int length = bytes.Length;
    char[] text = new char[length];
    for (int i = 0; i < length; i++)
        text[i] = (char)bytes[i];

    return new string(text);
}
```

Una vez finalizadas estas clases con sus respectivos métodos que se encargan de manejar las señales de protocolo UDP, tenemos que instanciar el servidor y asignar los eventos necesarios. Para esto, dentro de *Program* en el método *StartServer()*, nos vamos al *if* en que se definen los eventos, cuando todavía no hay una conexión establecida, y agregamos los siguiente:

```
UdpSocketServer udpServer = new UdpSocketServer();
udpServer.DataReceived += udpServer_DataReceived;
```

Con la primera línea estamos instanciando el servidor UDP (en el puerto por defecto 5123), y luego le asignamos el evento cuando reciba datos, que tiene que contener lo siguiente:

```
void udpServer_DataReceived(object sender, DataReceivedEventArgs e)
{
    string receivedMessage = UdpSocketServer.BytesToString(e.Data);

    if (receivedMessage == "IP")
    {
        // Creates a response and assigns it to the DataReceivedEventArgs.ResponseData
        // property, so that it will be automtically sent to client.
        string response = "IP/"+IPAddress;
        e.ResponseData = System.Text.Encoding.UTF8.GetBytes(response);
    }
}
```

Si llega una señal del broadcast de la aplicación móvil, se ejecuta este evento y primero lo que hace es filtrar si el mensaje contiene el string “IP”, de este modo eliminamos cualquier ruido que llegue al puerto (muy improbable).

Nota: este mensaje que contiene “IP” es previamente enviado con ese string específico, por la aplicación móvil (este desarrollo lo veremos más adelante).

Finalmente observamos que la placa responde el mensaje enviando: “*IP/”+IPAddress*. Esto quiere decir que la aplicación móvil recibirá un string nuevamente con el filtro “IP”, una separación por “/”, y luego el IP del servidor, que estaba alojado en la variable *IPAddress*.

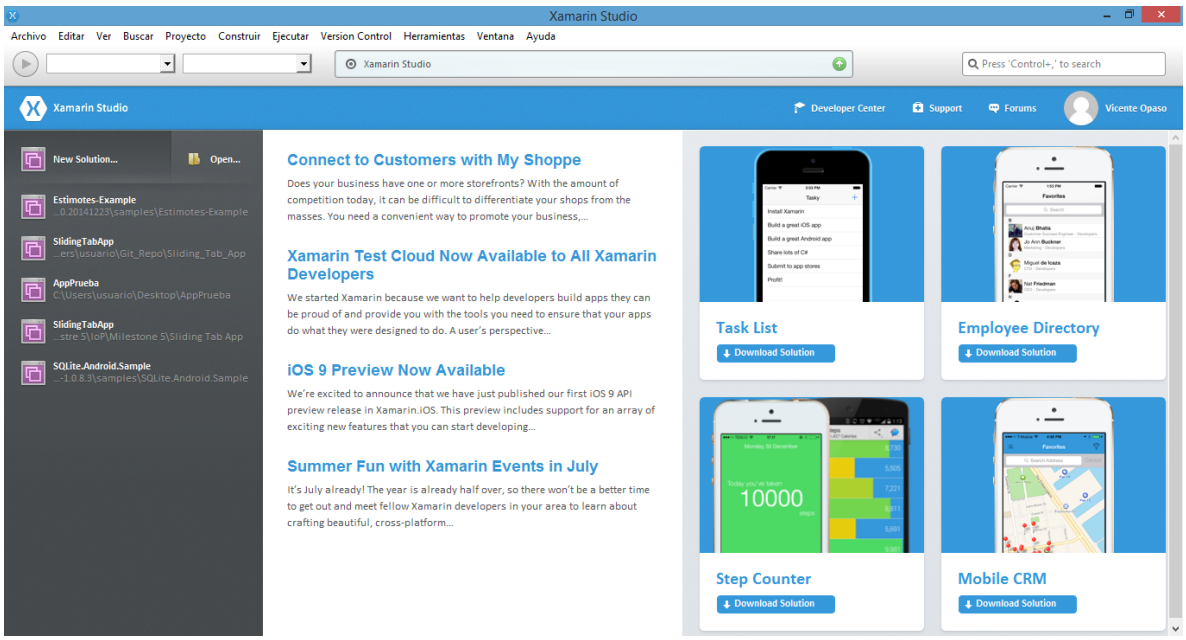
Una vez finalizada esta etapa, queda terminado el programa del microcontrolador y proseguiremos a desarrollar la aplicación en Xamarin Android.

Paso 4: Iniciar proyecto en Xamarin

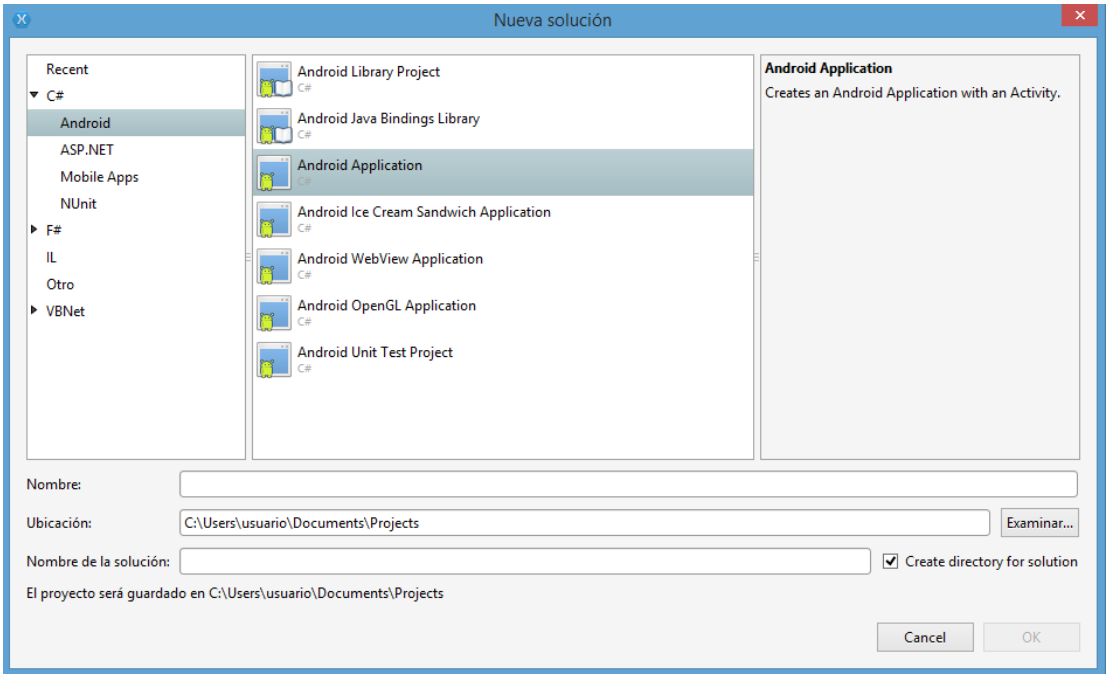
4.1) Nuevo proyecto Xamarin Android

A continuación, vamos a explicar cómo desarrollar la aplicación móvil que se instalará en un dispositivo Android y que permitirá controlar por WiFi los sensores y actuadores conectados a la placa Gadgeteer.

Para iniciar, abrimos el programa Xamarin Studio. Obtendremos una imagen como la siguiente:

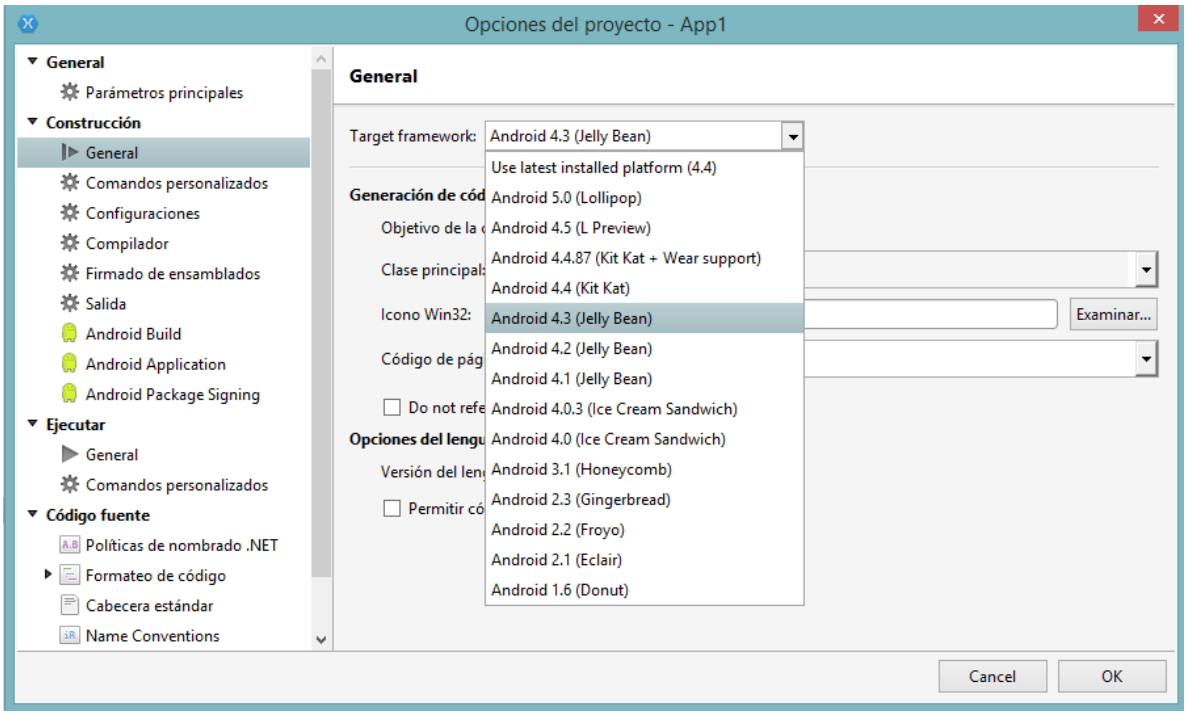


Luego en el costado izquierdo vamos a *New Solution*.
Nos aparecerá una ventana como la siguiente:



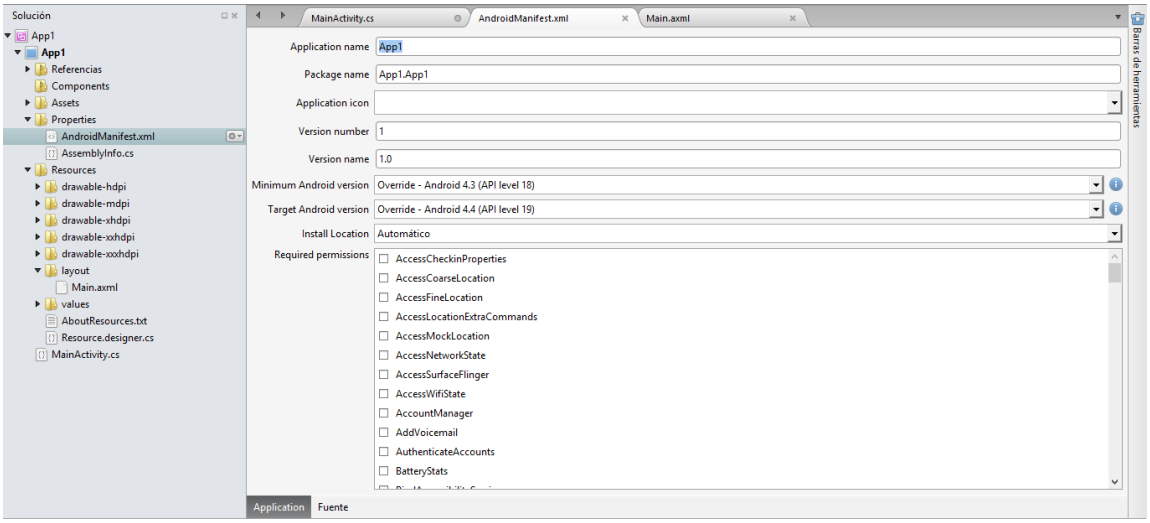
Aquí en el costado izquierdo, dentro de la sección *C#* seleccionamos *Android*. Luego en la ventana principal nos aparecerán varias opciones, y seleccionamos *Android Application*. Le colocamos nombre a la aplicación, en este caso le pondremos *App1*, luego le damos la dirección de almacenamiento deseada y luego pulsamos OK.

Una vez iniciada la solución, en la barra de menús, nos vamos a *Proyecto -> Opciones de la App1* (o el nombre de tu aplicación), luego aparecerá una ventana como la siguiente:



En el panel izquierdo en la sección de *Construcción*, seleccionamos *General*. Aquí desplegamos las opciones de *Target framework* y seleccionamos *Android 4.3 (Jelly Bean)*. Pulsamos *OK*.

Luego de vuelta en la ventana principal, nos vamos al panel del costado izquierdo y navegamos en las carpetas de la aplicación y buscamos *Properties*, luego hacemos doble click en *AndroidManifest.xml* y nos aparecerá un nuevo archivo en la ventana principal. Una vez aquí, vamos donde dice *Minimum Android versión* y seleccionamos *Override – Android 4.3 (API level 18)*.



Atención: En este tutorial trabajaremos con un Galaxy Note 3, que contiene Android 4.3, por eso seleccionamos estas opciones. Si se quiere trabajar con otra versión de Android, entonces cambiar estas configuraciones.

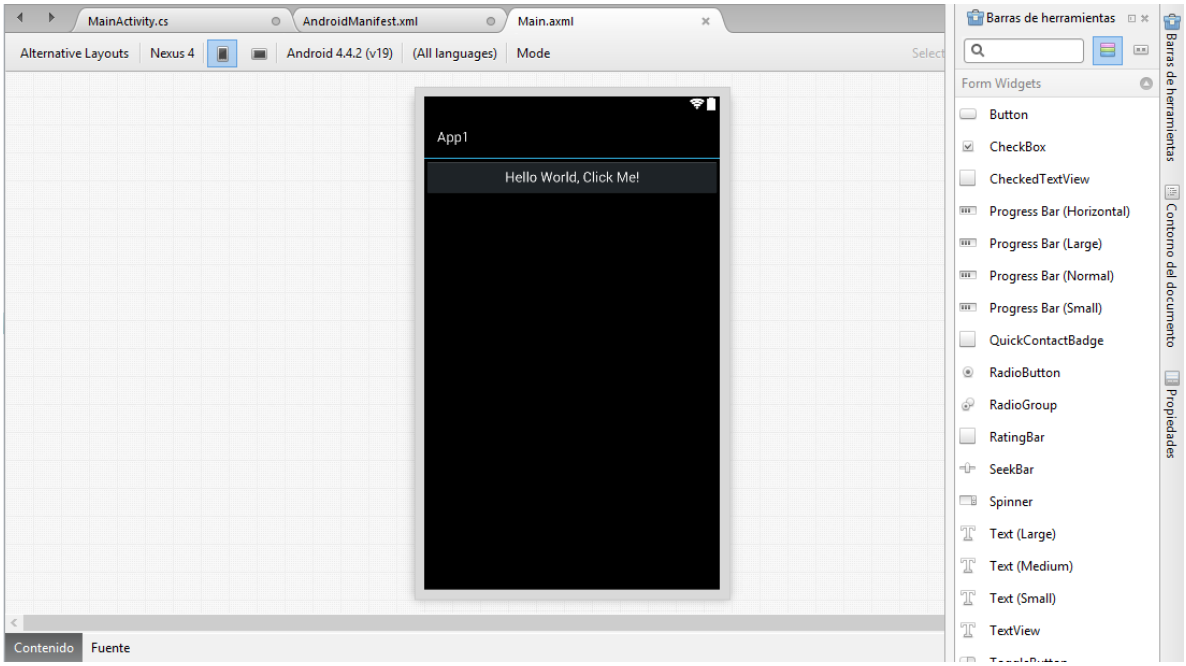
Una vez hecho estos cambios en las opciones de la aplicación y en el *AndroidManifest*, entonces nos vamos a la pestaña *MainActivity*.
Aquí borramos el código innecesario del código de ejemplo (la aplicación siempre inicia con una App sencilla de ejemplo). Entonces partimos con lo siguiente:

```
[Activity (Label = "App1", MainLauncher = true, Icon = "@drawable/icon")]
public class MainActivity : Activity
{
    protected override void OnCreate (Bundle bundle)
    {
        base.OnCreate (bundle);

        SetContentView (Resource.Layout.Main);
    }
}
```

4.2) Componentes gráficos

Primero, partiremos haciendo la parte gráfica de la aplicación. Navegamos en las carpetas de la solución y nos vamos a *Resources -> layout -> Main.axml*. Nos aparecerá el contenido de la pantalla principal con que se inicia la aplicación. Aquí vemos también la barra de herramientas para poder agregar widgets y objetos al contenido de la pantalla.



Borramos el botón *Hello World* de ejemplo, suprimiéndolo gráficamente o borrándolo desde su código de fuente.

En el sector inferior de la ventana, seleccionamos *Fuente* y nos aparecerá el código de fuente de la pantalla *Main*.

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3      android:orientation="vertical"
4      android:layout_width="fill_parent"
5      android:layout_height="fill_parent">
6      <Button
7          android:id="@+id/myButton"
8          android:layout_width="match_parent"
9          android:layout_height="wrap_content"
10         android:text="@string/hello" />
11 </LinearLayout>
```

Podemos borrar el botón de ejemplo suprimiendo la sección *<Button …. />* que en este caso sería borrar desde la línea 6 a la 10.
Luego en vez del botón inicial agregamos otros componentes y cambiamos un poco sus propiedades. Estos son sus códigos:

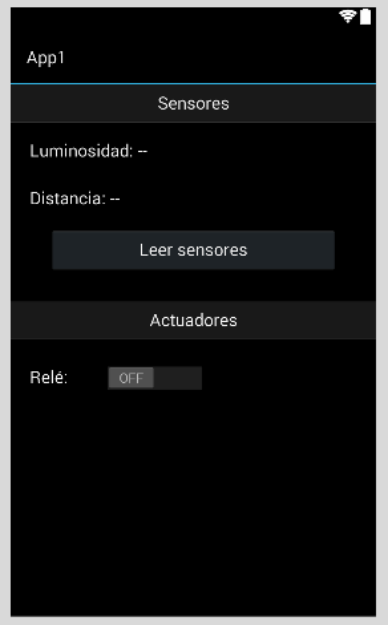
```
<TextView
    android:text="Sensores"
    android:textAppearance="?android:attr/textAppearanceMedium"
    android:layout_width="match_parent"
    android:layout_height="40dp"
    android:id="@+id/textSensors"
    android:gravity="center"
    android:background="@android:drawable/dark_header" />
<TextView
    android:text="Luminosidad: --"
    android:textAppearance="?android:attr/textAppearanceMedium"
    android:layout_width="match_parent"
    android:layout_height="40dp"
    android:id="@+id/textLight"
    android:gravity="center_vertical"
    android:layout_marginLeft="20dp"
    android:layout_marginRight="20dp"
    android:layout_marginTop="10dp" />
<TextView
    android:text="Distancia: --"
    android:textAppearance="?android:attr/textAppearanceMedium"
    android:layout_width="match_parent"
    android:layout_height="40dp"
    android:id="@+id/textDistance"
    android:gravity="center_vertical"
    android:layout_marginLeft="20dp"
    android:layout_marginRight="20dp"
    android:layout_marginTop="10dp" />
<Button
    android:text="Leer sensores"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:id="@+id/buttonSensors"
    android:layout_marginLeft="40dp"
    android:layout_marginRight="40dp"
    android:layout_marginTop="10dp" />
```

Estos componentes agregados serían la parte de sensores de la aplicación, en donde se desplegará la información de los sensores, es decir, el porcentaje de luminosidad y la distancia leída en

centímetros, además de incorporar el botón que hará la petición para leer los sensores de la placa. Luego se incorporan los componentes de actuadores, que será donde se controlará el relé para prender o apagar una luz.

```
<TextView
    android:text="Actuadores"
    android:textAppearance="?android:attr/textAppearanceMedium"
    android:layout_width="match_parent"
    android:layout_height="40dp"
    android:id="@+id/textActuators"
    android:gravity="center"
    android:background="@android:drawable/dark_header"
    android:layout_marginTop="30dp" />
<TextView
    android:text="Relé: "
    android:textAppearance="?android:attr/textAppearanceMedium"
    android:layout_width="match_parent"
    android:layout_height="40dp"
    android:id="@+id/textRelay"
    android:gravity="center_vertical"
    android:layout_marginLeft="20dp"
    android:layout_marginRight="20dp"
    android:layout_marginTop="20dp" />
<Switch
    android:layout_width="wrap_content"
    android:layout_height="40dp"
    android:id="@+id/switchRelay"
    android:layout_marginTop="-40dp"
    android:layout_marginLeft="100dp" />
```

Cabe destacar que los atributos se han modificado para que quede una buena imagen gráfica en la pantalla *Main*, como cambiar los márgenes, alinear los textos, etc. Si volvemos a la sección *Contenido*, que es donde se ve gráficamente la pantalla, podemos agregar estos *TextView*, botones y otros componentes desde la barra de herramientas a un costado derecho, y los atributos de estos componentes se pueden cambiar desde *Propiedades* o desde el código mismo como lo hicimos recién. Una vez incorporados estos componentes y sus propiedades, tendremos una imagen de lo que será la pantalla principal de la aplicación.



Atención: Hay que fijarse en los identificadores que les ponemos a los componentes que hemos agregado, con la definición a*ndroid:id="@+id/…."*. En este caso los identificadores que iremos a utilizar para hacer uso de ellos en la Actividad serán *textLight*, *textDistance*, *buttonSensors* y *switchRelay*.

Ahora volvemos a la pestaña *MainActivity.cs* y definimos algunas variables de la Actividad:

```
[Activity (Label = "App1", MainLauncher = true, Icon = "@drawable/icon")]
public class MainActivity : Activity
{
    string IP = "0.0.0.0";
    string port = "80";
    TextView textLight;
    TextView textDistance;
    Button buttonSensors;
    Switch switchRelay;
    private const int listenPort = 5123;

    protected override void OnCreate (Bundle bundle)
    {
        base.OnCreate (bundle);
        SetContentView (Resource.Layout.Main);

        //Creating objects that represents the components of the screen
        textLight = FindViewById<TextView>(Resource.Id.textLight);
        textDistance = FindViewById<TextView>(Resource.Id.textDistance);
        buttonSensors = FindViewById<Button>(Resource.Id.buttonSensors);
        switchRelay = FindViewById<Switch>(Resource.Id.switchRelay);
    }
}
```

Agregamos la variable *IP*, que contendrá la IP del servidor alojado en la placa, el puerto a utilizar y además el puerto correspondiente al broadcast UDP, alojado en la variable *listenPort*. Luego creamos los objetos que representan los componentes de la pantalla. Como se observa, dentro del método *OnCreate()* se definen estos objetos para luego poder manipular la parte gráfica con ellos.

4.3) Eventos y métodos

Lo primero que haremos es crear un *enum* para trabajar de forma ordenada con los dispositivos a manejar, que serán los sensores o el relé. El *enum* se define fuera de la clase y lo llamaremos *Device*.

```
enum Device { Sensors, Relay };

[Activity (Label = "App1", MainLauncher = true, Icon = "@drawable/icon")]
public class MainActivity : Activity
…
```

Ahora empezaremos con la lógica de la aplicación. Para eso, volvemos al método *OnCreate()* y creamos el evento para el botón de leer los sensores:

```
using System.Threading;
...

botonSensores.Click += delegate
{
    botonSensores.Text = "...";
    ThreadPool.QueueUserWorkItem(o => Request(Device.Sensors));
};
```

Este evento se ejecuta cuando se hace click en este botón. Lo que hace es correr el método *Request()*, que lo definiremos luego. Este método *Request()* lo corre en un Background Thread para

no paralizar la aplicación, por lo tanto, cuando hagamos la petición con el botón podremos seguir utilizando y manipulando la pantalla mientras el programa procesa la conexión y el intercambio de datos.

Similarmente se define el evento del switch del relé:

```
switchRelay.Click += delegate
{
    ThreadPool.QueueUserWorkItem(o => Request(Device.Relay));
};
```

Una vez hecho esto, procedemos a definir el método *Request()*:

```
using System.Net;
…

void Request (Device dev)
{

    string url = "http://"+ IP +":"+ port +"/";
    string code;
    switch (str) {
    case Device.Sensors:
        url = url + "Sensors";
        code = performRequest(url);
        RunOnUiThread(() => {
            buttonSensors.Text = "Leer sensores";
            if (code!=null)
            {
                string[] values = code.Split('/');
                //Here we show the values obtained in the reading sensor
                textLight.Text = "Luminosidad: " + values[0] + "%";
                textDistance.Text = "Distancia: " + values[1] + " cm.";
            }
        });
        break;
    case Device.Relay:
        url = url + "Relay";
        code = performRequest(url);
        break;
    }
}
```

Este método depende del valor que le pasemos, dependiendo si queremos leer los sensores o controlar el relé. En el primer caso, la *url* será con el path de *Sensors*, y se ejecutará el método *performRequest(url)* que lo definiremos luego. Este método hará la petición al servidor y se almacenará la respuesta en la variable *code*. Luego para hacer los cambios gráficos, tenemos que volver al UI Thread, y aquí ejecutamos los cambios relacionados con los datos obtenidos.

Atención: se hace un nuevo *string[] valores* que almacena los datos separados por un "/", ya que en el programa Gadgeteer habíamos definido que la respuesta era de la forma *{luz}/{distancia}*. Por ejemplo, si la variable *code* es: "63.8/24" entonces la variable *valores* será ["63.8","24"]

Luego en el caso que se quiera controlar el relé, solamente haremos la petición ejecutando el método *performRequest(url)*, ya que no nos interesa una respuesta dentro de la aplicación Android, sino que solamente una respuesta en la aplicación Gadgeteer que pueda encender o apagar el relé (permita o no el paso de corriente). Entonces el método que hace la petición es el siguiente:

```
using Java.Net;
using Java.IO;
using System.IO;
...

public String performRequest(String link)
{
    URL url;
    HttpURLConnection urlConnection = null;

    try
    {
        url = new URL(link);
        urlConnection = (HttpURLConnection)url.OpenConnection();
        urlConnection.RequestMethod = "GET";
        int responseCode = (int)urlConnection.ResponseCode;

        Stream stream = urlConnection.InputStream;
        InputStreamReader isr = new InputStreamReader(stream);
        BufferedReader reader = new BufferedReader(isr);

        String response = "";
        String line;
        while ((line = reader.ReadLine()) != null)
        { response += line; }

        reader.Close();
        urlConnection.Disconnect();
        return response;
    }

    catch (Exception e) { var a = e.StackTrace; }

    finally
    {
        if (urlConnection != null) urlConnection.Disconnect();
    }
    return null;
}
```

Lo que hace básicamente es tratar de hacer la petición mediante el método GET, hacia la *url* establecida anteriormente. En caso exitoso retornará la respuesta dada por la variable *response*.

4.4) IP y UDP broadcast

Ahora necesitamos hacer que al inicio de la aplicación se haga un UDP broadcast, es decir, se manda una señal UDP a todas las IP de la red local, para que le llegue al servidor alojado en la placa y éste responda con su dirección IP, de modo de poder almacenar en la variable *IP* de nuestro programa, el IP correspondiente a la placa y así poder hacer las peticiones para leer los sensores o manejar el relé.

Entonces procedemos a instanciar un *UdpClient*. Dentro del método *OnCreate()* vamos al final, luego de los eventos de click, y colocamos lo siguiente:

```
using System.Text;
…
try
{
    UdpClient client = new UdpClient();
    byte[] sendbuf = Encoding.ASCII.GetBytes("IP");
    IPEndPoint ep = new IPEndPoint(IPAddress.Broadcast, listenPort);
    client.Send(sendbuf, sendbuf.Length, ep);
    client.Close();
}
catch (Exception e)
{ }
```

Aquí definimos un *UdpClient*, codificamos la información a enviar, definimos un *IPEndPoint* para hacer un broadcast con puerto *listenPort*, que en el caso de este tutorial es 5123. Luego se envía la información y finalmente se cierra el *UdpClient*.

En el broadcast, se envía el mensaje almacenado en *sendbuf*, mediante el protocolo UDP a todas las IP posibles. La variable *sendbuf* en este caso es el string "IP", que correspondía al filtro que verificaba el servidor de la placa, cuando recibía datos.
De este modo, al hacer el broadcast, en algún momento le llegará el mensaje a la placa y ésta ejecutará su respuesta.

Nota: Esta petición UDP para obtener el IP al estar en el método *OnCreate()*, la estamos ejecutando al inicio de la aplicación y dentro de un *try-catch*, de modo que si falla la conexión o el dispositivo móvil no estaba conectado a la misma red local que la placa, el broadcast fallará pero la aplicación no se caerá y se iniciará de todos modos, pero la comunicación con la placa no funcionará. Si por alguna razón se requiere hacer nuevamente la petición del IP de la placa, basta con cerrar la aplicación e iniciarla nuevamente.

Una vez hecho el envío, tenemos que ser capaces de escuchar la respuesta que nos mande la placa, por lo tanto hay que inicializar un servidor que esté escuchando. Primero en *OnCreate()* vamos a donde hicimos el broadcast, justo antes del *try*, y colocamos lo siguiente:

```
ThreadPool.QueueUserWorkItem(o => StartListener());
```

De este modo llamamos al método *StartListener()* que crearemos a continuación, que se encargará de escuchar las señales UDP entrantes. Esto lo hacemos en un Background Thread para que el programa no se congele mientras se ejecuta este método.

Entonces el método *StartListener()* contiene lo siguiente:

```
void StartListener()
{
    bool done = false;

    UdpClient listener = new UdpClient(listenPort);
    IPEndPoint groupEP = new IPEndPoint(IPAddress.Any,listenPort);

    try
    {
        while (!done)
        {
            //Waiting the broadcast
            byte[] bytes = listener.Receive(ref groupEP);

            //Broadcast received
            string msg = Encoding.ASCII.GetString(bytes,0,bytes.Length);
            if (msg.Split('/').Length >= 2 & msg.Split('/')[0] == "IP")
            {
                IP = msg.Split('/')[1];
            }
        }

    }
    catch (Exception e)
    { }
    finally
    {
        listener.Close();
    }
}
```

El programa queda escuchando las señales entrantes, y cuando la placa responda adecuadamente con su IP, en el *while* se ejecutará el código de recepción del mensaje y guardará la IP de la placa Gagdeteer en la variable global *IP* de esta aplicación.

Finalmente ya construida la aplicación, podremos continuar al siguiente paso, que será probar el programa completo.

Errores comunes:

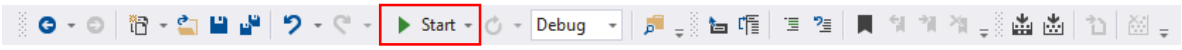
- En la sección 4.1 puede suceder que al tratar de cambiar las configuraciones, no aparezca la versión Android que queremos (API level). Para esto tendremos que descargar el API level desde el SDK Manager. Nos vamos a *Herramientas -> Open Android SDK Manager*, y se nos abrirá un ventana para poder instalar nuevos paquetes. Aquí buscamos la API level requerida de la versión Android y la instalamos.

Paso 5: Probar programa Gadgeteer – Xamarin Android

5.1) Debug Gadgeteer

Para correr y probar el programa realizado haremos Debug con los dos programas de la placa y del dispositivo. Esto quiere decir que depuraremos en modo de prueba y corrección de errores. Así veremos cómo funciona nuestro programa realizado.

Para empezar, conectamos físicamente a la placa todos los componentes Gadgeteer a utilizar y verificamos que estas conexiones son las mismas que existen en nuestro programa. Luego conectamos el módulo *USB Client SP* a la placa mediante un cable USB, para darle energía a la placa. De este modo, una vez conectado al computador, en la ventana principal nos vamos a la barra de opciones y con la opción *Debug* seleccionada, pulsamos *Start*.



Ahora empezará a compilar y a traspasar el programa a la placa. Luego de unos segundos se iniciará el programa.

En la ventana emergente *Output* se irá actualizando todo lo que pasa en el Debug. Si observamos el código desarrollado, se definieron líneas con el comando *Debug.Print("mensaje")*, de este modo se lleva registro en tiempo real de lo que va sucediendo en el programa.

A continuación, para correr la aplicación Android y utilizarla, primero debemos esperar que la placa se conecte a la red WiFi. Para saber esto, en el *Output* se indicará este mensaje una vez que esté conectada correctamente. Además se indicará la IP asignada y el puerto.

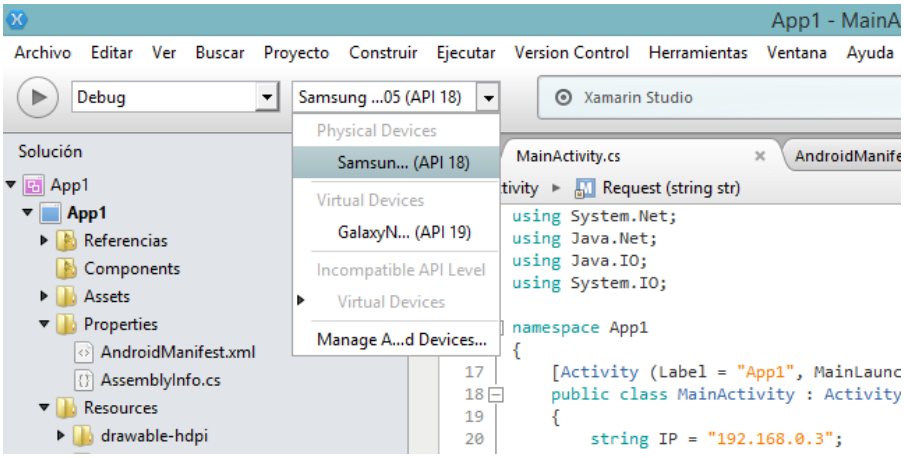
Una vez terminado este proceso, procedemos a correr la aplicación móvil.

5.2) Debug Xamarin

Primero que nada, hay que saber que hay más de una forma de hacer Debug para probar la aplicación Android. Por ejemplo, se puede hacer mediante un emulador que desplegará una aproximación gráfica del dispositivo virtualmente en la pantalla. Esta emulación depende del dispositivo que se quiera emular, ya que varía según cada versión de Android. Se puede configurar un dispositivo virtual que quedará almacenado dentro de *Virtual Devices*. Para más información de Debug ver: http://developer.xamarin.com/guides/android/deployment,_testing,_and_metrics/

También existen otras formas de Debug, como por ejemplo, correr el programa directamente al dispositivo conectado al computador. Para efectos de este tutorial, utilizaremos esta última opción.

Para proceder al Debug, primero conectamos el dispositivo al computador mediante su cable USB (o medio de conexión). Luego en la barra superior nos debería aparecer un nuevo dispositivo:



Debajo de *Physical Devices* seleccionamos nuestro dispositivo recién conectado.

Finalmente nos aseguramos que esté colocada la opción *Debug*, y luego pulsamos el botón *Start*.

Entonces el programa compilará y se iniciará en el dispositivo conectado.

5.3) Prueba de programa completo

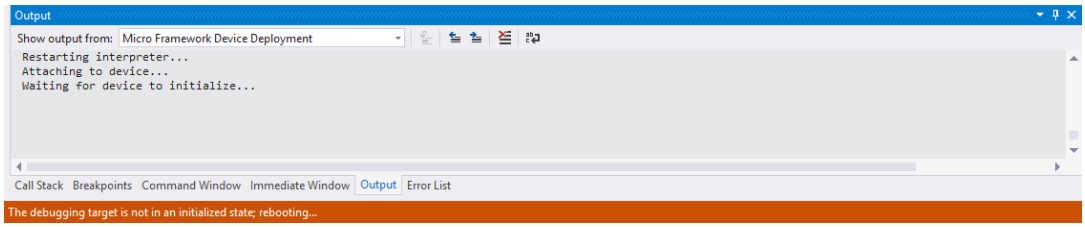
Finalmente asegurándonos que la placa esté andando, con el WiFi conectado a la red y que la aplicación móvil este corriendo correctamente, entonces procedemos a utilizar la aplicación.

Primero, con el dispositivo Android nos conectamos a la misma red WiFi que la placa. Luego pulsamos el botón *Leer sensores*. Mientras procesa debiera cambiarse el texto del botón a "…". Luego de unos instantes, en el *Output* del programa Gadgeteer debiera indicar que se recibió una petición y debiera ejecutar la respuesta correctamente. Luego en el dispositivo se actualizarán los textos de luminosidad y distancia, indicando los datos obtenidos por los sensores.

Del mismo modo, si prendemos o apagamos el switch de la aplicación, entonces se prenderá o apagará el led del relé que indica si está encendido o no (si se le conecta una ampolleta, esta se encenderá y apagará).

Errores comunes:

- En la sección 5.1 pueden suceder varios problemas que nos impidan debuggear correctamente. Por ejemplo se puede dar que una vez compilada la solución, el programa se quede pegado antes de correrlo en la placa. En general hay que tener paciencia y probar nuevamente:



Pero esto también se puede deber a otros problemas: por ejemplo, una vez que conectamos la placa al computador, esta puede hacer sonidos de conexión y desconexión constantemente en una especie de intermitencia. Para este tipo de problemas de energía, la solución es alimentar la placa con el módulo USB Client DP, que además de conectado al computador, habrá que conectarlo a un enchufe (ver errores comunes de sección 1.3), además de no olvidarse de hacer el cambio en las conexiones del programa Gadgeteer.

- En la sección 5.2 si no aparece la opción de nuestro dispositivo conectado debajo de *Physical Devices*, entonces verificar que el computador reconoce correctamente el dispositivo. Si aún sigue sin aparecer, entonces podría deberse a que tenemos que instalar el API level necesario (ver errores comunes de la sección 4.1)

- En la sección 5.3 si no funcionan las peticiones cuando apretamos los botones ya sea de leer sensores o del relé, entonces una de las causas posibles de este problema es que la petición no se esté haciendo correctamente a la IP de la placa Gadgeteer. Para ello hay que verificar en el método *Request()* el valor de la variable *url* (se puede hace debuggeando) y ver qué IP está utilizando para hacer las peticiones, o simplemente ver en tiempo de ejecución el valor que se asigna a la variable *IP* y verificar que realmente sea la IP de la placa.

Consideraciones finales

Finalmente hemos terminado y probado la aplicación completamente!

Si se siguieron adecuadamente los pasos del tutorial, en la prueba final debiera funcionar todo correctamente. Si siguen persistiendo los errores, entonces revisar los errores comunes de cada sección y buscar en internet toda la información que puedan para solucionar su problema!

Y por sobre todo, mucho ánimo y paciencia!

El código de ejemplo de este tutorial se puede descargar aquí:
https://github.com/vaopaso/Gadgeteer---Xamarin-Android.git