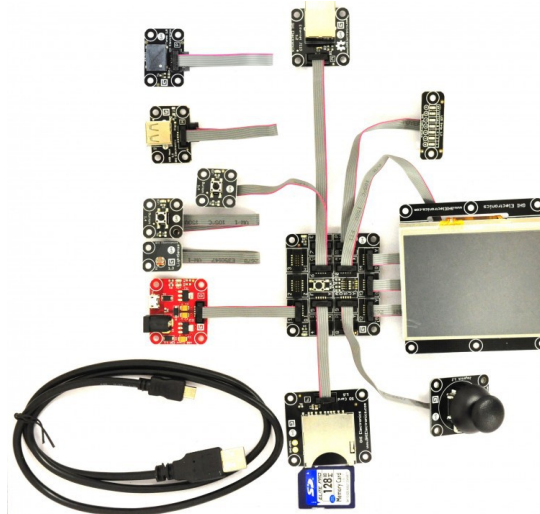


.NET Gadgeteer for Beginners

April 17, 2015

Getting Started



A “plug and play” for electronics, with a style!

*** Draft, in-progress document ***



Licensed under Creative Commons Share Alike 4.0

www.creativecommons.org/licenses/by-sa/4.0/

Table of Contents

1. About the Book.....	3
1.1. Intended Audience.....	3
1.2. Disclaimer	3
2. Introduction.....	4
2.1. .NET Micro Framework.....	4
2.2. Visual Studio.....	5
3. Helpful Notes for Your Gadgets.....	6
4. LED Blinking, the easy way.....	7
4.1. Setting up.....	7
4.2. Coding the Software.....	11
4.3. Troubleshooting.....	15
5. LED Blinking, the right way.....	16
6. LED Blinking, using a module.....	17
7. Namespace.....	20
8. Reading a Digital Input.....	22
5. Traffic Light!.....	26
9. Drawing Simple Shapes.....	29
9.1. Glide.....	30
10. Drawing Images.....	31
11. Drawing Fonts.....	34
12. Reading an Analog Input.....	36
13. Memory Storage.....	38
14. Networking.....	40
14.1. Network Settings.....	40
14.2. Pinging the Device.....	42
14.3. A Known Bug!.....	44
14.4. Network Controlled Devices.....	44
15. Keeping it Tidy!.....	49
16. Gadgeteering the non-Gadgeteer!.....	50
16.1. Accessing the Pins.....	54
16.2. Not using Mainboards.....	54
17. Visual Basic.....	55
18. Prototype to Production.....	56
18.1. The Hardware.....	56
18.2. The Software.....	56
18.3. .NET Gadgeteer in Production.....	56

1. About the Book

1.1. Intended Audience

This book is for beginners wanting to get started on .NET Gadgeteer. No prior knowledge is necessary, but some basic programming knowledge is helpful. There is a great deal of information in this book for both hobbyists and engineers.

1.2. Disclaimer

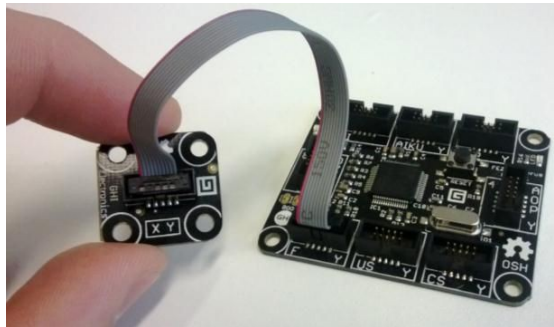
This is a free book. Use it for your own knowledge and at your own risk. GHI Electronics is not responsible for any damage or loss caused by the use of this book or by any information provided. There is no guarantee any information in this book is valid.

2. Introduction

***** Draft, in-progress document *****

.NET Gadgeteer is a rapid development platform that utilizes mainboards and plug-and-play modules to build electronic projects, without the need for an electronics background. Take a temperature logger application for example, this requires a mainboard, SD card module and temperature module. As for the software, all the necessary drivers are included, significantly reducing development time. The few lines of code needed for this example can be coded by anyone with minimal software experience.

These specifications are introduced and maintained by Microsoft and the community for standardizing the connections between mainboards and modules.



A .NET Gadgeteer module on the left, a .NET Gadgeteer mainboard on the right. A cable connects sockets between the module and the mainboard.

2.1. .NET Micro Framework

.NET Gadgeteer software works on top of the .NET Micro Framework (NETMF) software. NETMF includes thousands of methods, some are borrowed from the full .NET framework. Meaning .NET Desktop developers already know how to program NETMF and .NET Gadgeteer products. NETMF also extends the standard framework with numerous hardware-related methods. This includes the ability to read and write to control pins allowing you to blink some lights or to check if a button is pressed. GHI Electronics also extends the framework with some exclusive features, such as using databases and USB Host.

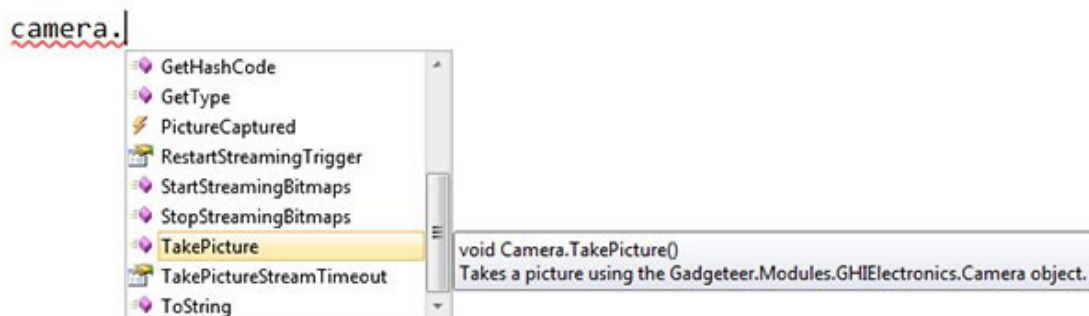
2.2. Visual Studio

.NET Gadgeteer developers will be using Microsoft's Visual Studio, one of the most popular tools, by hobbyists and professionals. Even the free version of Visual Studio will work with .NET Gadgeteer.

One of the many benefits of using Visual Studio is its IntelliSense. As you type, IntelliSense understands what you might be trying to do and displays boxes with suggestions. That's why we ship .NET Gadgeteer products without any programming manuals. It's that FEZ!

Learn about the history behind **Fast and Easy (FEZ)** here:
www.ghielectronics.com/technologies/fez

Let's say a Camera Module is being used. All you need to do is type the word "camera" and hit "period" on the keyboard to see a list of supported methods available in the included drivers.



The beauty of object-oriented programming at its finest.

Deploying programs to the device is done through a simple USB cable. What sets NETMF and .NET Gadgeteer apart from other entry level electronic circuits is its ability to debug code. Developers can step in code, pause executions, check variables and add breakpoints.

We've barely scratched the surface of what .NET Gadgeteer with NETMF and Visual Studio can do. Let's put it all to the test and enjoy watching some blinking lights!

3. Helpful Notes for Your Gadgets

As .NET Gadgeteer uses the latest professional tools on the market, a proper software installation is necessary. Beginners should stick to the exact software versions listed on the GHI Electronics' website or within this book.

To get started, install the exact software listed on GHI Electronics' support page:

<https://www.ghielectronics.com/support/gadgeteer>

The installation steps will include:

1. Microsoft Visual Studio (IDE and the compiler)
2. Microsoft NETMF Project System (plugin for Visual Studio to allow deploying/debugging)
3. Microsoft NETMF SDK (framework libraries)
4. Microsoft's .NET Gadgeteer (graphical designer and project templates)
5. GHI Electronics' SDK (drivers for mainboards and modules plus GHI Electronics' exclusive libraries)

This book's examples are made with modules available in the tinker kits from GHI Electronics; however, any other .NET Gadgeteer hardware should work with minor changes.

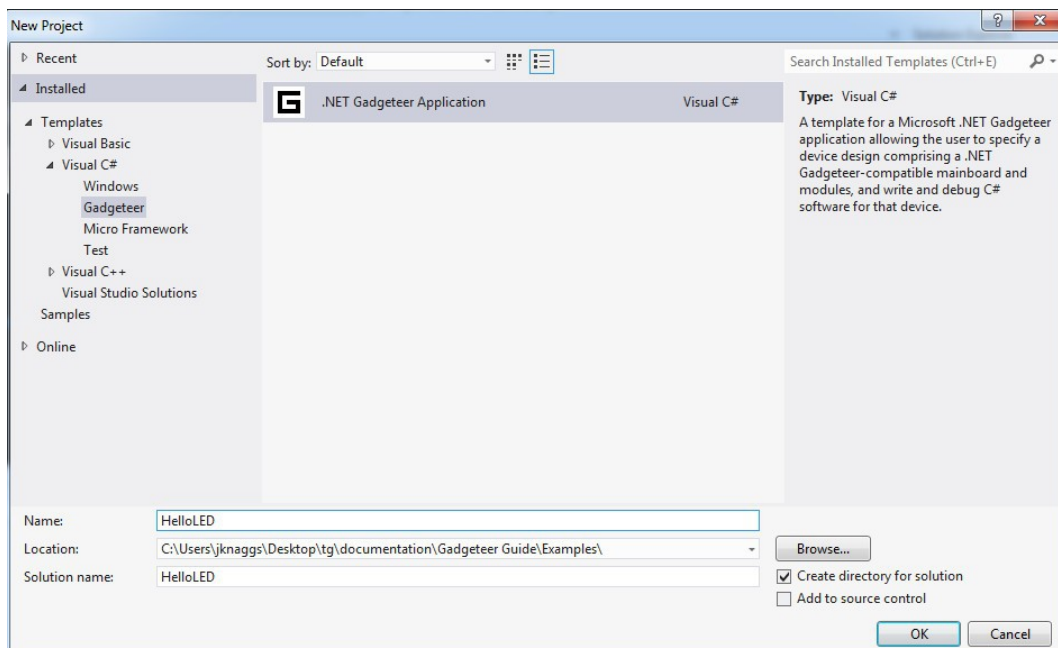
4. LED Blinking, the easy way

Now, let's develop and run a small application to make sure everything is setup properly. We'll blink the LED that is found on every mainboard and print a debug string back to Visual Studio's output window.

Tip: Please note that there is a troubleshooting section at the end of this chapter.

4.1. Setting up

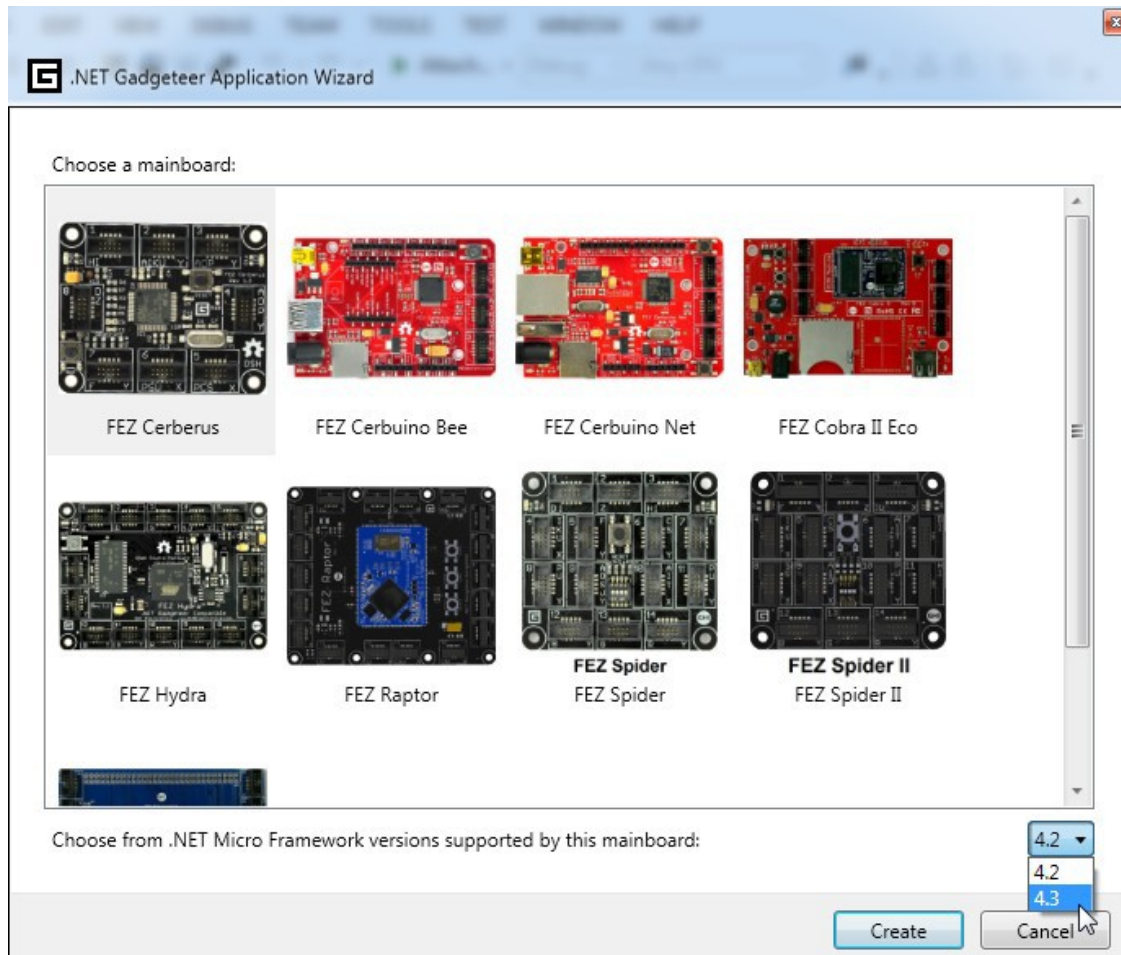
Run Visual Studio and start a new project. Click on "FILE>New Project...". You will have the option of using Visual Basic or C#. This book will focus on C# but there is a Visual Basic chapter near the end.



IMPORTANT: Use HelloLED for the project name. This is very important! I will explain why later.

The next step is to select the proper mainboard. Make sure to also select the latest version of .NET Micro Framework if this option is available. Some mainboards may support multiple

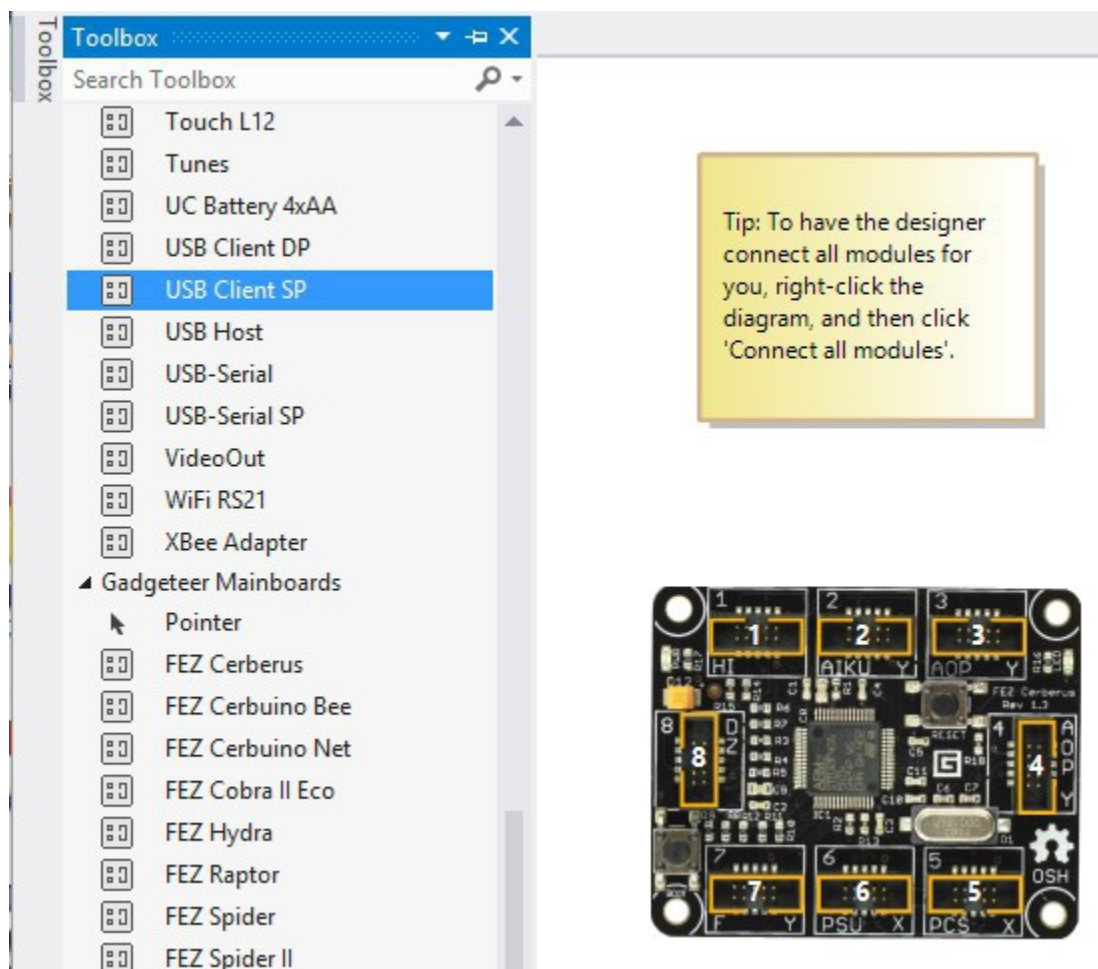
versions of NETMF and we want to always use the latest if possible.



Click the Create button and the .NET Gadgeteer Visual Designer's window will now appear with the mainboard selected.

Tip: Highlight or click on any mainboard/module in the design surface and hit F1. This will show a useful help page about that particular mainboard/module.

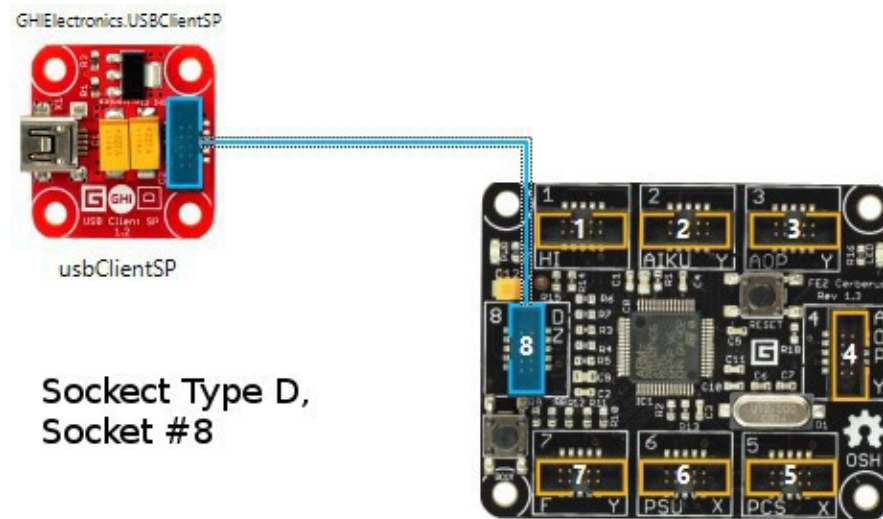
Click the Toolbox tab to show all the available modules and mainboards.



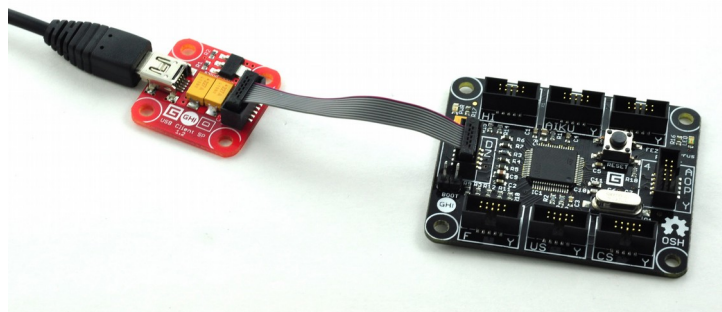
We want to start by adding a power module to the design. This is also the module used for debugging/deploying over USB. The power modules are always red and you should only have one power module added in any design.

Find the module in your kit and locate its match in the designer. In my case, it is USB Client SP module. Add the module in the designer by dragging it in or by double-clicking.

We are now ready to connect the module to the mainboard. You can right-click anywhere in the designer area and click “connect all modules” or click on the socket on the module you want to connect and the mainboard will automatically highlight the correct sockets. If you noticed, the module's socket is labeled with D and on the mainboard the proper socket will also be labeled D.



Now go back to your actual boards and make the same connection.



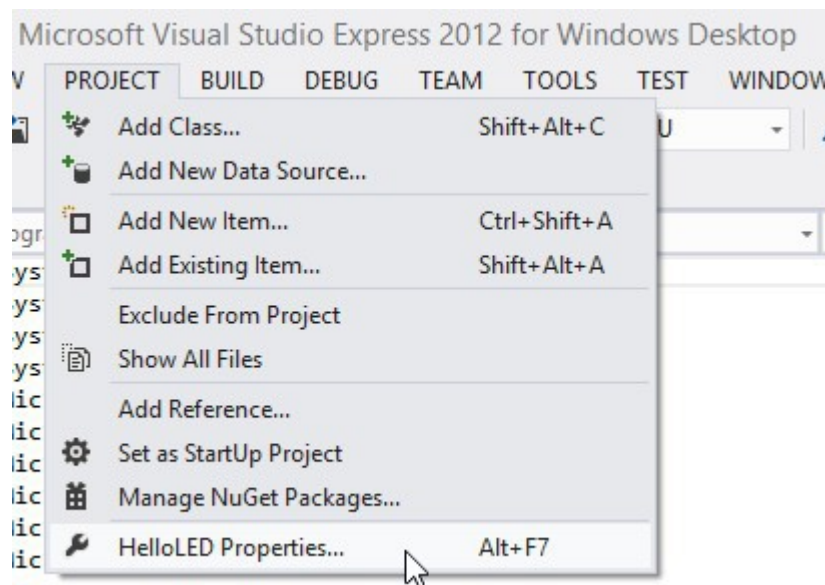
Connect the power module to your computer using a standard USB Cable. Observe the red light on the power module (red module).

4.2. Coding the Software

Now that you got this far, you deserve to have fun and enjoy .NET Gadgeteer. Remember to save your project if you haven't done so already.

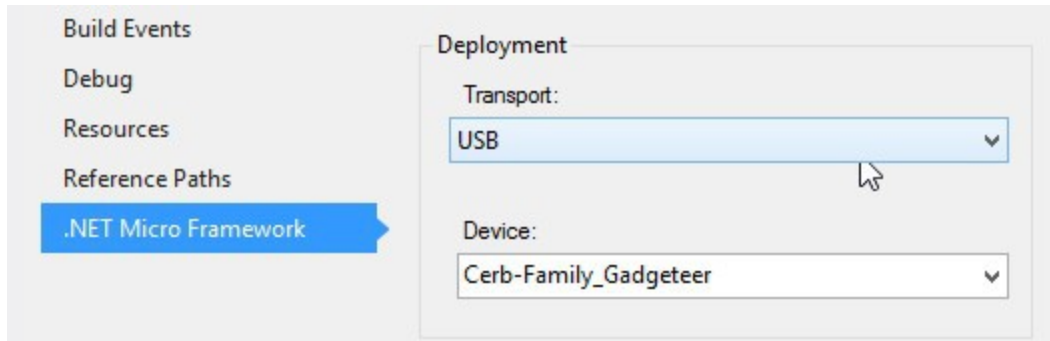
Locate the Solution Explorer window. You will find two C# files in there. One is auto-generated by the designer surface, where you added the modules. Do not modify this file. Now click Program.cs to view it. In the code, you will see a "ProgramStarted" method and inside of it there is a debug statement. Everything in green is considered a comment.

Now, how does Visual Studio know which board to access, in case you have multiple. It also needs to know how the board is connected (USB in our case). To set these settings, go to the project properties.

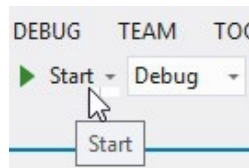


By the way, my project is named HelloLED but yours can be anything else.

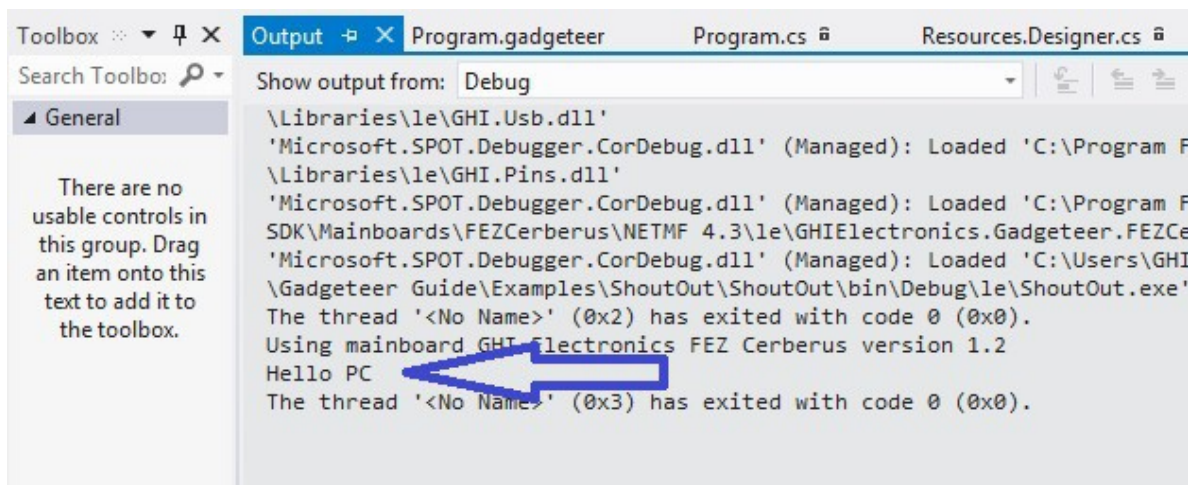
Click on the .NET Micro Framework tab and check the transport is set to USB and find your device under devices. If you do not see the device then it is not plugged in properly.



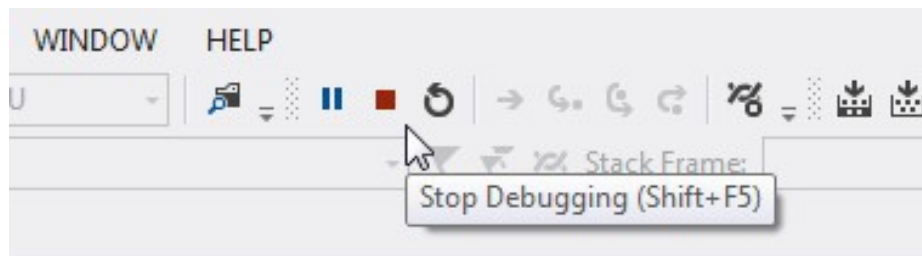
Now we are ready to deploy our code to the device. Click the start button or simply hit F5.



This compiles the code and deploys it to the device to be ran. You will see a lot of activity on the output window, then the program will load and run. The debug print is generated by the mainboard but the message is sent back to Visual Studio over the USB cable to be shown in the output window.



This is a good start but we don't have any blinking LEDs yet! Lets fix that. Start by stopping the program by clicking the button or Shift + F5.



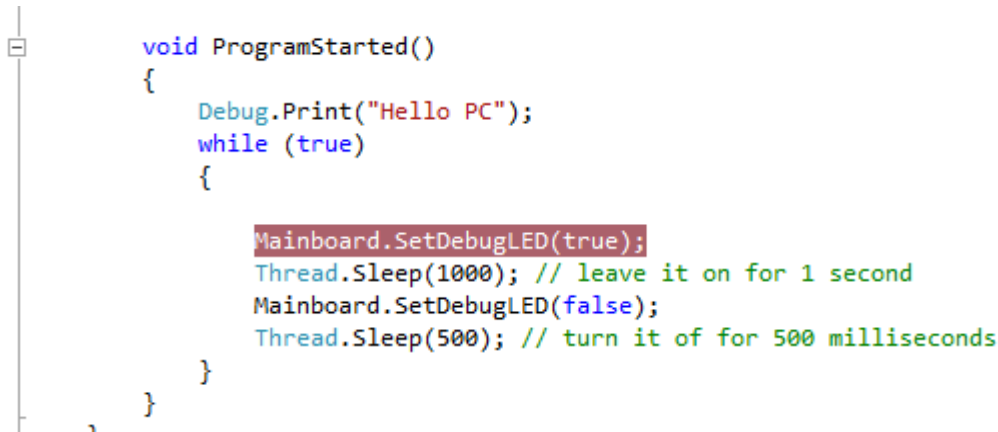
Go back to Program.cs and edit it with the following code to blink the no-board LED. By the way, LED stands for a light emitting diode.

```
using System;
using GT = Gadgeteer;
using GTM = Gadgeteer.Modules;

namespace HelloLED
{
    public partial class Program
    {
        void ProgramStarted()
        {
            Debug.Print("Hello PC");
            while (true)
            {
                Mainboard.SetDebugLED(true);
                Thread.Sleep(1000); // leave it on for 1 second
                Mainboard.SetDebugLED(false);
                Thread.Sleep(500); // turn it of for 500 milliseconds
            }
        }
    }
}
```

Run the program again and the LED on the mainboard will be blinking.

You can now enjoy the debugging features. While the LED is blinking, set the cursor over the first SetDebugLED line and hit F9 key. This should make that line red and will stop the program at that line.



```
void ProgramStarted()
{
    Debug.Print("Hello PC");
    while (true)
    {
        Mainboard.SetDebugLED(true);
        Thread.Sleep(1000); // leave it on for 1 second
        Mainboard.SetDebugLED(false);
        Thread.Sleep(500); // turn it off for 500 milliseconds
    }
}
```

The LED should be off at this exact moment. The line to turn the LED on has not executed just yet. Now hit the F10 key for a single step and note the program and the LED.

4.3. Troubleshooting

Do not panic if the LED is not blinking by now. There are two common things that can go wrong. First, is the USB driver. Did you see the device's name when going to the project settings in Visual Studio? If not, check the device manager. The driver should install manually when it's installing the GHI Electronics' SDK. Check the installation steps on the support page. You can also point windows directly to the driver, located in the GHI Electronics folder under program files, or where you selected to install the GHI SDK.

Second, is when the device is not loaded with the latest software. When this is the problem, you should see a message on the output windows indicating firmware mismatch. The device (the mainboard) has a resident software, called firmware that always lives on it. This is not the application you develop and load onto the device. The firmware is developed and provided by the mainboard manufacture. Consult with the mainboard manuals on how to update the firmware.

Tip: GHI Electronics provides a utility software called FEZ Config to help with many tasks, including firmware updates.

Finally, the software running the mainboard may have very tight loops making it difficult for Visual Studio to latch to the device. Pressing the reset button at the same time as deploying from Visual Studio helps. In extreme cases, the device need to be set in TinyBotter mode to erase the application. This should not be needed if the system is properly programmed by using events without using long tight loops and no sleep.

5. LED Blinking, the right way

In the previous chapter, we used a simple infinite loop to blink the LED. Using infinite loops is an old practice that shouldn't be used with a modern system like .NET Gadgeteer. The correct alternative is to start a timer and then toggle the LED every time the timer ticks.

```
using System;
using GT = Gadgeteer;
using GTM = Gadgeteer.Modules;

namespace HelloLED
{
    public partial class Program
    {
        // This method is run when the mainboard is powered up or reset.
        void ProgramStarted()
        {
            GT.Timer timer = new GT.Timer(300); // Create a timer
            timer.Tick += timer_Tick; // Run the method timer_tick when the timer ticks

            timer.Start(); // Start the timer
        }

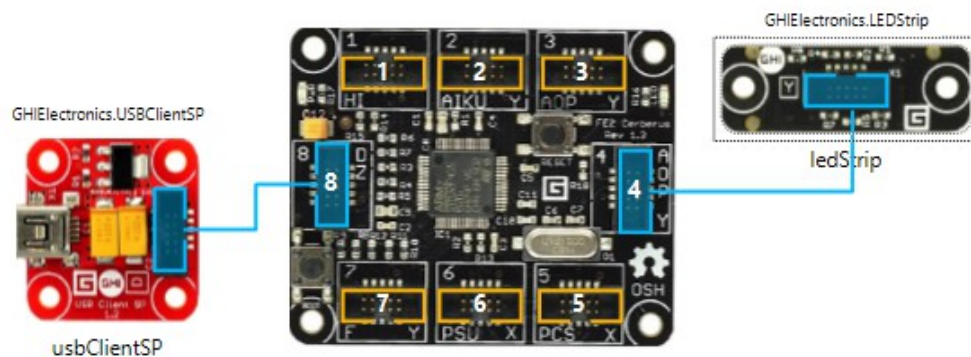
        bool state = false;
        void timer_Tick(GT.Timer timer)
        {
            state = !state; // Invert the state (to blink the LED)
            Mainboard.SetDebugLED(state); // Set the LED to state
        }
    }
}
```

This method may seem overly complex for blinking an LED (which is true), but complete applications do a lot more than blinking an LED. I will give you a good example. How would you blink the LED once every 765 milliseconds and debug print the word 'hello' once every 567 milliseconds? Try to accomplish this with a single loop. It will be extremely difficult. Now, try it using individual timers.

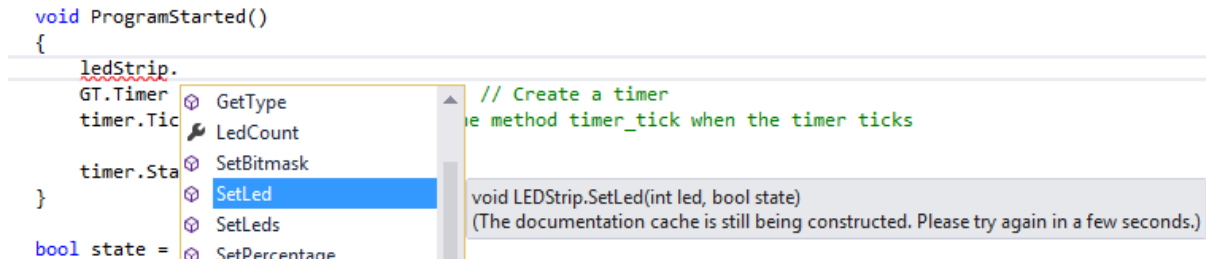
Another concern solved by using timers is that the ProgramStarted method needs to complete and return. Adding an infinite loop inside of it will prevent it from returning, causing the .NET Gadgeteer core to not work properly.

6. LED Blinking, using a module

We are about to plug in the first module and use it, congratulations. This chapter will be using the LED Strip module. The first thing to do would be to go back to the designer surface and add the LED Strip module, or any other LED module you may have.



The instant name of the LED Strip module is shown right under the module. In this case it is ledStrip. Go back to the code and type the module name to inspect what methods are available through this module.



Going through the list, you should see the SetLed method. It should be very easy to modify the code we had earlier to blink one of the LEDs on the module instead of the mainboard's LED.

```
using System;
using GT = Gadgeteer;
using GTM = Gadgeteer.Modules;
using Gadgeteer.Modules.GHIElectronics;

namespace HelloLED
{
    public partial class Program
    {
        // This method is run when the mainboard is powered up or reset.
        void ProgramStarted()
        {
            GT.Timer timer = new GT.Timer(300); // Create a timer
            timer.Tick += timer_Tick; // Run the method timer_tick when the timer ticks

            timer.Start(); // Start the timer
        }

        bool state = false;
        void timer_Tick(GT.Timer timer)
        {
            state = !state; // Invert the state (to blink the LED)
            ledStrip.SetLed(1, state); // Set the LED to state
        }
    }
}
```

Now, let's experiment with other available methods, like TurnAllLedsOff and TurnAllLedsOn. Can you get these methods to blink all LEDs together?

```
using System;
using GT = Gadgeteer;
using GTM = Gadgeteer.Modules;
using Gadgeteer.Modules.GHIElectronics;

namespace HelloLED
{
    public partial class Program
    {
        // This method is run when the mainboard is powered up or reset.
        void ProgramStarted()
        {
            GT.Timer timer = new GT.Timer(300); // Create a timer
            timer.Tick += timer_Tick; // Run the method timer_tick when the timer ticks

            timer.Start(); // Start the timer
        }

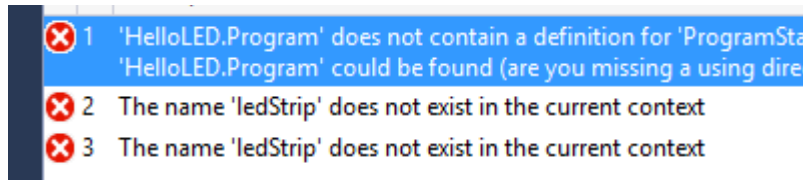
        bool state = false;
        void timer_Tick(GT.Timer timer)
        {
            state = !state; // Invert the state (to blink the LED)
            if (state)
                ledStrip.TurnAllLedsOn();
            else
                ledStrip.TurnAllLedsOff();
        }
    }
}
```

7. Namespace

Namespaces are a way to organize and control code in large projects. It groups code sections under a name space. When that space needs to be used, the “using” statement is utilized. All examples provided earlier show multiple using statements at the beginning of the code. Do not worry if you do not completely understand this, just remember to add the using statements as shows in the example code for now.

What is important to understand now is the namespace of your actual program. When you create a project called MyProject, Visual Studio will automatically set the namespace to MyProject. If you manually modify the namespace of your code to something else then your .NET Gadgeteer project will no longer compile. This will most likely happen when you copy code from this book or the internet.

Tip: This is why in an earlier chapter it was important for you to name the project HelloLED. Try to go back to the last example and change the namespace to something else, like HelloLEDs. The error will look something like this image:

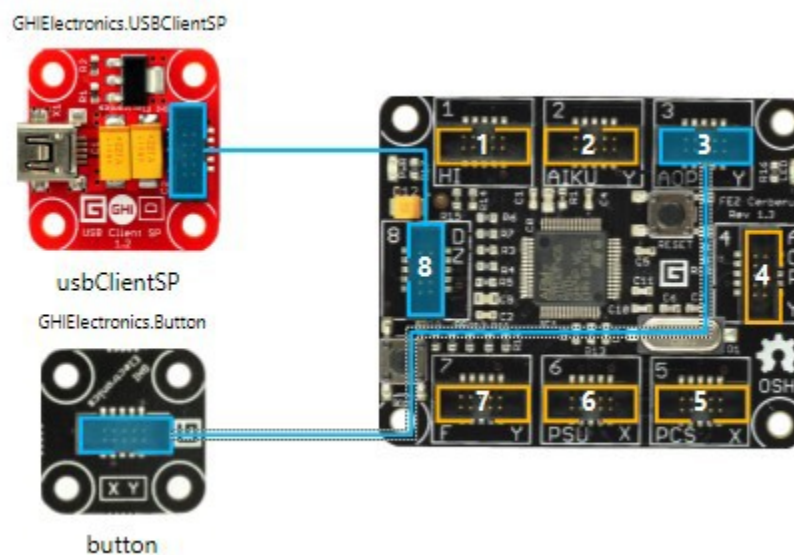


Always make sure to correct the namespace when copying examples from this book. For the sake of simplicity, all following examples will be built with a project named GadgeteerBook.

8. Reading a Digital Input

Lets connect a button and use it to control the LED found on the mainboard. Start a new project and name it GadgeteerBook, as explained in the namespace chapter. Note that everything is case sensitive when it comes to programming in C#, Gadgteerbook will not work.

Here's our design of the hardware, set yours up the same way:



Remember that you can hit F1 while highlighting one of the modules for the available methods. Lets do that for the button.

GTM.GHIElectronics.Button Program.gadgeteer Program.cs*

URL: C:\Users\GHI Developer\AppData\Local\Temp\GTM.GHIElectronics.But

Properties

Name	Description
Pressed	Whether or not the button is pressed.
IsLedOn	Whether or not the LED is currently on or off.
Mode	Gets or sets the LED's current mode of operation.

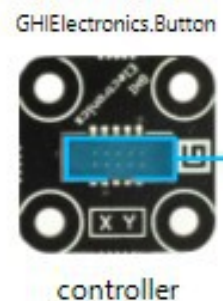
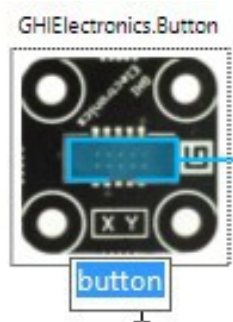
Methods

Name	Description
TurnLedOn	Turns on the LED.
TurnLedOff	Turns off the LED.
ToggleLED	Turns the LED off if it is on and on if it is off.

Events

Name	Description
ButtonReleased	Raised when the button is released.
ButtonPressed	Raised when the button is pressed.

In previous examples, we used the modules with their default name. In this example, we will change the button name from button to controller. Click on the name right under the module and change as showing below.



Lets start with one of the earlier examples that we used to blink the on-board LED on the mainboard and slightly modify it to only blink if the LED is pressed.

```
using System;
using GT = Gadgeteer;
using GTM = Gadgeteer.Modules;
```

```
using Gadgeteer.Modules.GHIElectronics;

namespace GadgeteerBook
{
    public partial class Program
    {
        // This method is run when the mainboard is powered up or reset.
        void ProgramStarted()
        {
            GT.Timer timer = new GT.Timer(300); // Create a timer
            timer.Tick += timer_Tick; // Run the method timer_tick when the timer ticks

            timer.Start(); // Start the timer
        }

        bool state = false;
        void timer_Tick(GT.Timer timer)
        {
            state = !state; // Invert the state (to blink the LED)
            if (controller.Pressed) // If button is pressed
            {
                Mainboard.SetDebugLED(state);
            }
        }
    }
}
```

While the code is correct, it does not take advantage of the button events. We can use less system resources and code by properly using events. If a button pressed event fired, we will start the blink timer. If the button was released, we will stop the timer.

```
using System;
using GT = Gadgeteer;
using GTM = Gadgeteer.Modules;
using Gadgeteer.Modules.GHIElectronics;

namespace GadgeteerBook
{
    public partial class Program
    {
        private GT.Timer timer = new GT.Timer(300); // Create a timer
        void ProgramStarted()
        {
            timer.Tick += timer_Tick; // Run the method timer_tick when the timer ticks
            controller.ButtonPressed += controller_ButtonPressed; // Button down event
            controller.ButtonReleased += controller_ButtonReleased; // Button up event
        }

        void controller_ButtonReleased(Button sender, Button.ButtonState state)
        {
            timer.Stop(); // Stop the timer
        }

        void controller_ButtonPressed(Button sender, Button.ButtonState state)
        {
            timer.Start(); // Start the timer
        }

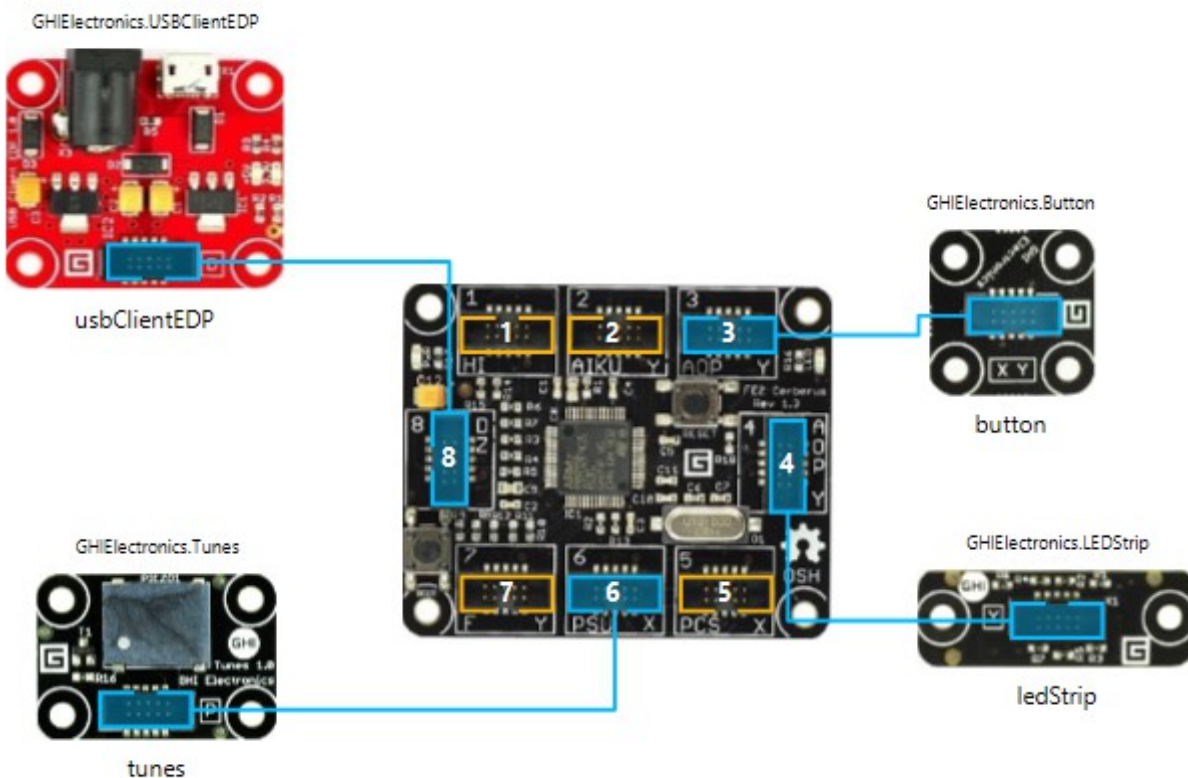
        bool state = false;
        void timer_Tick(GT.Timer timer)
        {
            state = !state; // Invert the state (to blink the LED)
            Mainboard.SetDebugLED(state);
        }
    }
}
```

Can you use the button to control the LEDs on a module? What about the button controlling the speed of blinking?

Hint: Change the timer speed (interval).

5. Traffic Light!

In this chapter, we will bring it all together, a button with LEDs and even add some sounds from the tunes module. Lets call this example a traffic light. When the button is pressed, the light will automatically become red to stop the vehicles while the tunes module makes a sound to allow pedestrians to cross safely!



The code properly uses two timers and never blocks the system with an infinite loop. Try to analyze and modify the program.

```
using System;
using System.Collections;
using System.Threading;
using Microsoft.SPOT;
using GT = Gadgeteer;
using GTM = Gadgeteer.Modules;
using Gadgeteer.Modules.GHIElectronics;

namespace GadgeteerBook
{
    public partial class Program
    {
        private enum LightState : uint
        {
            GREEN = 1 + 2, // 00 000 11
            YELLOW = 4 + 8 + 16, // 00 111 00
            RED = 32 + 64 // 11 000 00
        }
        int SecondsCounter = 0;
        int WalkingCounter = 0;
        GT.Timer CarTimer = new GT.Timer(1000);
        GT.Timer PedTimer = new GT.Timer(1000);

        void ProgramStarted()
        {
            CarTimer.Tick += CarTimer_Tick;
            PedTimer.Tick += PedTimer_Tick;
            button.ButtonPressed += ButtonPressed;
            CarTimer.Start();
        }

        public void ButtonPressed(Button sender, Button.ButtonState state)
        {
            // Stop the light at RED for cars
            CarTimer.Stop();
            ledStrip.SetBitmask((uint)LightState.RED);
            SecondsCounter = 0;
            // Allow pedestrians to walk for 5 seconds
            WalkingCounter = 5;
            PedTimer.Start();
        }

        void PedTimer_Tick(GT.Timer timer)
        {
            if (WalkingCounter-- < 0)
            {
                // Last round
                PedTimer.Stop();
                CarTimer.Start();
                // Make a longer sound
                tunes.Play(4000, 300);
            }
            else
            {
                tunes.Play(4000, 50); // Make short sounds
            }
        }

        private void CarTimer_Tick(GT.Timer timer)
        {
            if (SecondsCounter < 4) // start with Red
            {
                ledStrip.SetBitmask((uint)LightState.RED);
            }
        }
    }
}
```

```
    }  
    else if (SecondsCounter < 7) // Red-> Green  
    {  
        ledStrip.SetBitmask((uint)LightState.GREEN);  
    }  
    else if (SecondsCounter < 8) // Green-> Yellow  
    {  
        ledStrip.SetBitmask((uint)LightState.YELLOW);  
    }  
    else // back to the beginning  
    {  
        SecondsCounter = 0;  
    }  
    SecondsCounter++; // Increment the seconds counter  
}  
}
```

Tip: If you are not interested in stepping through the code (debugging), it maybe faster to just load the program without debugging. This is accomplished by pressing Shift + F5.

9. Drawing Simple Shapes

One of the great features of .NET Gadgeteer is in its graphics support. This graphical drawing feature is inherited from the core of .NET Micro Framework, which supports drawing simple shapes and also supports BMP, JPG and GIF images. Not only that, full font support is also part of the package. More on fonts in the next chapter.

We will also be using Threads instead of Timers in this example. Note how the thread has an infinite loop but it also has a sleep that allows the rest of the system to run properly.

```
using System;
using System.Threading;
using GT = Gadgeteer;
using GTM = Gadgeteer.Modules;
using Gadgeteer.Modules.GHIElectronics;

namespace GadgeteerBook
{
    public partial class Program
    {
        void ProgramStarted()
        {
            displayN18.SimpleGraphics.DisplayRectangle(
                GT.Color.Magenta, 2, GT.Color.Black, 1, 1, 100, 100);
            Thread Bouncer = new Thread(BouncerLoop);
            Bouncer.Start();
        }
        int x = 50, y = 50; // position
        int dx = 4, dy = 6; // speed and direction
        void BouncerLoop()
        {
            while (true) // Infinite loop
            {
                x += dx;
                y += dy;
                displayN18.SimpleGraphics.DisplayEllipse(
                    GT.Color.Green, 1, GT.Color.Black, x, y, 8, 8);
                // Bounce back!
                if (x < 15 || x > 85)
                    dx *= -1;
                if (y < 15 || y > 85)
                    dy *= -1;
                Thread.Sleep(30); // Sleep for few milliseconds
            }
        }
    }
}
```

Tip: If Visual Studio failed to deploy an application after loading this program, simply click the reset button on the mainboard as soon as you click run on visual studio. This is especially the case with displays that use the SPI bus, like the Display N18 module.

9.1. Glide

For more advanced graphical application with beautiful user interfaces, GHI Electronics offer a free and open source library, called Glide. Each screen, called a window, is constructed and rendered from a simple XML text. It supports many components, like buttons and lists.

The website also hosts a window designer for glide. Drag and drop what you need and the website will generate the XML text.

Learn more at <https://www.ghielectronics.com/glide>

10. Drawing Images

As mentioned earlier, the core .NET Micro Framework supports BMP, JPEG and GIF image formats. This support is optional to the device manufacturers. Smaller devices with little resources may or may not support one or more of the image formats. Check the device manual to determine what image formats are supported on your device.

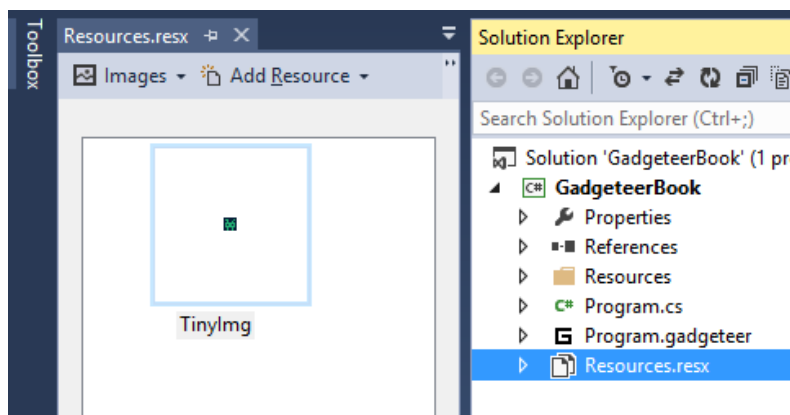
It is also important to note that images require a lot of memory. A simple 100x100 pixel image will need about 40,000 bytes. This is not a problem for devices with a large amount of memory but some .NET Gadgeteer devices have a very limited memory. These limited devices will typically run everything fine but will struggle with graphics.

To keep the demo universal, we will use a tiny image that is 8x8 pixels in this demo. You can use a larger image if your device has plenty of memory.



The next step is to add this bitmap as a resource in our program. It can also be loaded from the SD card but we will cover memory cards later.

Find Resources.resx in the Solution Explorer and double-click it. Now drag the image into the resource.



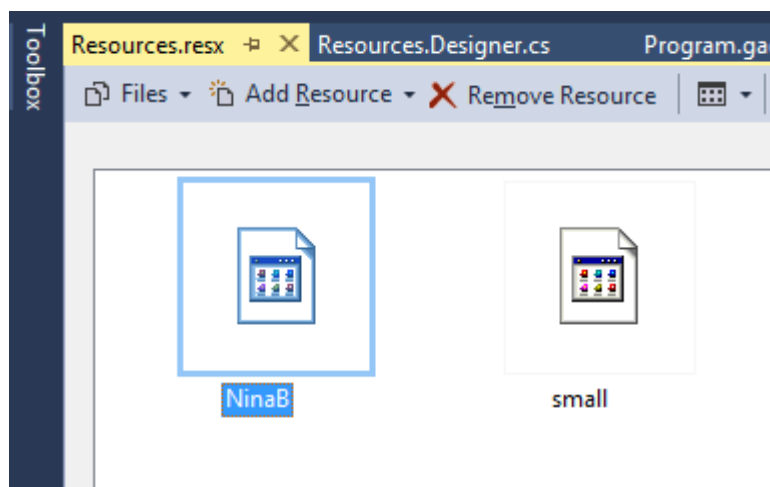
Lets make this cute monster run on the screen!

```
using System;
using System.Threading;
using Microsoft.SPOT;
using GT = Gadgeteer;
using GTM = Gadgeteer.Modules;
using Gadgeteer.Modules.GHIElectronics;

namespace GadgeteerBook
{
    public partial class Program
    {
        void ProgramStarted()
        {
            // Draw a border
            displayN18.SimpleGraphics.DisplayRectangle(
                GT.Color.Magenta, 2, GT.Color.Black, 1, 1, 100, 100);
            Thread Bouncer = new Thread(BouncerLoop);
            Bouncer.Start();
        }
        int x = 50, y = 50; // position
        int dx = 4, dy = 6; // speed and direction
        void BouncerLoop()
        {
            // Get the bitmap resource
            Bitmap monster = new Bitmap(Resources.GetBytes(
                Resources.BinaryResources.TinyImg), Bitmap.BitmapImageType.Bmp);
            displayN18.SimpleGraphics.AutoRedraw = false; // Do not redraw
            while (true)
            {
                // Erase the old monster
                displayN18.SimpleGraphics.DisplayRectangle(
                    GT.Color.Black, 0, GT.Color.Black, x, y, 8, 8);
                x += dx; y += dy;
                displayN18.SimpleGraphics.DisplayImage(monster, x, y);
                // Draw to the screen
                displayN18.SimpleGraphics.Redraw();
                // Bounce back!
                if (x < 15 || x > 85)
                    dx *= -1;
                if (y < 15 || y > 85)
                    dy *= -1;
                Thread.Sleep(30);
            }
        }
    }
}
```

11. Drawing Fonts

One of the great features of .NET Micro Framework is in the font support. You can take almost any font of any size and convert it to a TinyFont and use it on any .NET Micro Framework device with graphics support. This book will not explain how to convert fonts but the default .NET Gadgeteer project will always include two fonts. Go back to the resource viewer, by clicking on Resources.resx. From the drop down menu select Files. This should show the two included fonts, NinaB and small.



To write to the screen, we would need to construct a Font object and then use that to draw on the screen.

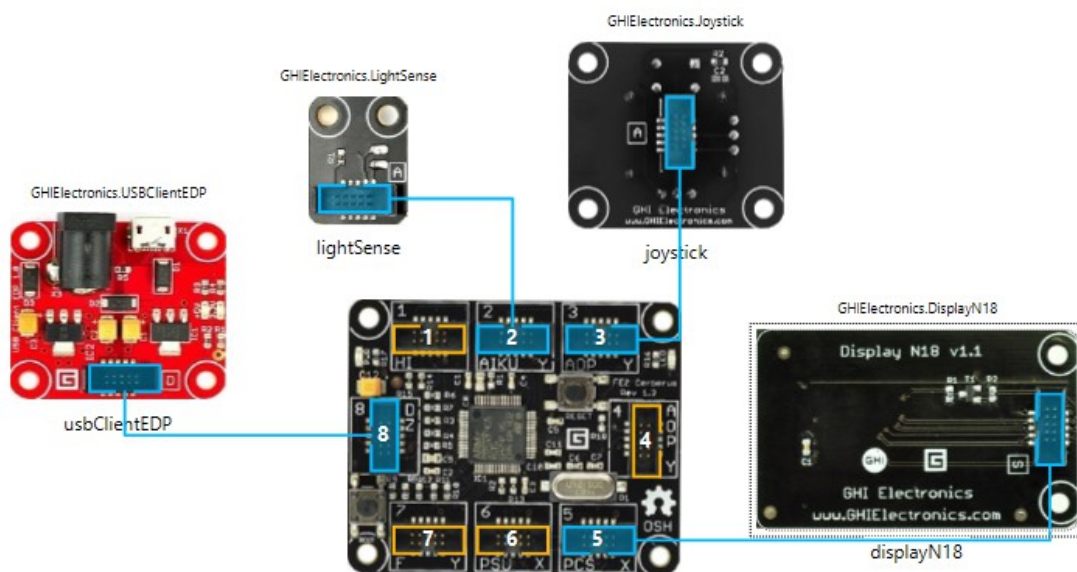
The example code is nearly identical to the one before, except it bounces the text “HI” inside the box.


```
using System;
using System.Threading;
using Microsoft.SPOT;
using GT = Gadgeteer;
using GTM = Gadgeteer.Modules;
using Gadgeteer.Modules.GHIElectronics;

namespace GadgeteerBook
{
    public partial class Program
    {
        void ProgramStarted()
        {
            // Draw a border
            displayN18.SimpleGraphics.DisplayRectangle(
                GT.Color.Magenta, 2, GT.Color.Black, 1, 1, 100, 100);
            Thread Bouncer = new Thread(BouncerLoop);
            Bouncer.Start();
        }
        int x = 50, y = 50; // position
        int dx = 4, dy = 6; // speed and direction
        void BouncerLoop()
        {
            // Get the bitmap resource
            Font MyFont = Resources.GetFont(Resources.FontResources.NinaB);
            displayN18.SimpleGraphics.AutoRedraw = false; // Do not redraw
            while (true)
            {
                // Erase the old text
                displayN18.SimpleGraphics.DisplayRectangle(
                    GT.Color.Black, 0, GT.Color.Black, x, y, 14, 14);
                x += dx; y += dy;
                displayN18.SimpleGraphics.DisplayText("HI", MyFont, GT.Color.Yellow, x, y);
                // Draw to the screen
                displayN18.SimpleGraphics.Redraw();
                // Bounce back!
                if (x < 15 || x > 85)
                    dx *= -1;
                if (y < 15 || y > 85)
                    dy *= -1;
                Thread.Sleep(30);
            }
        }
    }
}
```

12. Reading an Analog Input

Digital inputs provide a logical state, either on or off. In code, the digital state is true and false as learned before. On the other hand, analog inputs read the voltage level on a pin. This can be a light sensor or a joystick. In this demo, we will show a simple graph on the screen representing both, a light sensor and movement from an analog joystick. The code can be modified to work with any other analog input.



If you are new to C#, this code has something new to learn. The value returned from the sensors is a number that can carry a fraction called float or double. Double has more accuracy than float. For example the value coming from the joystick can be anywhere from -1.0 to 1.0, a possible output is 0.34 for example. These numbers are represented differently internally inside the processor from even numbers, called integers. Since handling integers is much faster on computers, they are only used when necessary. I will give you an example. When counting people who have entered a building, you will never have 1.54 of a person. This is also important to note when doing math. What is 5 divided by 2? Actually the answer is 2 not 2.5 if you are using integers.

In the following example we get the value from one of the sensors, which is double. We then run some simple math to scale the value to show nicely on the screen. The the vale is converted to an integer, through something called casting. This is required as drawing on the screen will only accept integers.

```

using System;
using System.Threading;
using Microsoft.SPOT;
using GT = Gadgeteer;
using GTM = Gadgeteer.Modules;
using Gadgeteer.Modules.GHIElectronics;
namespace GadgeteerBook
{
    public partial class Program
    {
        void ProgramStarted()
        {
            // Draw a border
            displayN18.SimpleGraphics.DisplayRectangle(
                GT.Color.Magenta, 2, GT.Color.Black, 1, 1, 100, 100);
            Thread GraphThread = new Thread(GraphLoop);
            GraphThread.Start();
        }
        void GraphLoop()
        {
            int x = 2;
            int LastLight = 0, LastJoyX = 0, LastJoyY = 0;
            while (true)
            {
                int Light = (int)(lightSense.ReadProportion() * 30) + 10; // Scale and cast
                displayN18.SimpleGraphics.DisplayLine(GT.Color.Blue, 1,
                    x - 1, LastLight, x, Light); // Draw
                LastLight = Light;

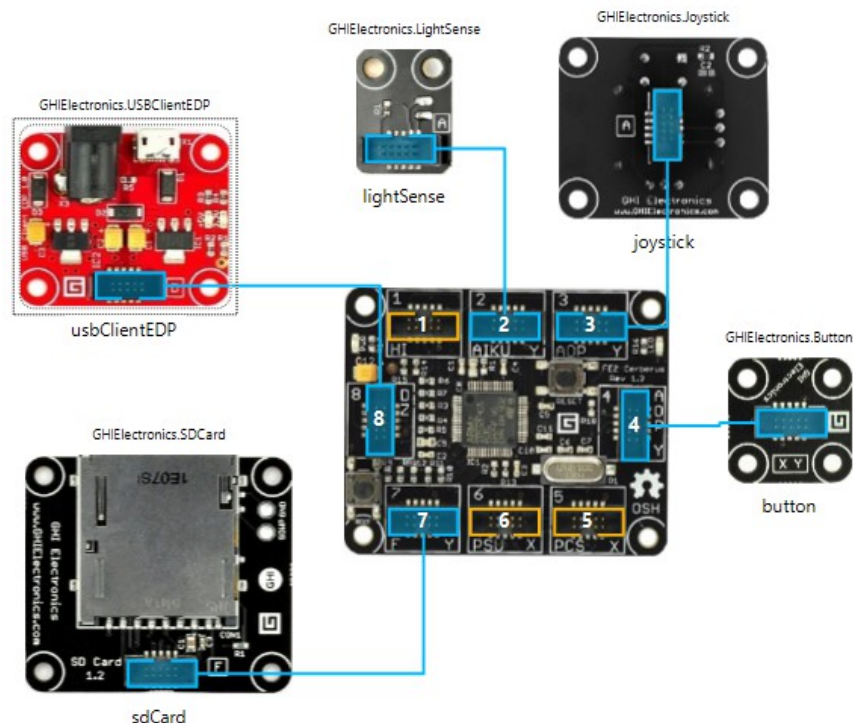
                int JoyX = (int)(joystick.GetPosition().X * 15) + 40; // Scale and cast
                displayN18.SimpleGraphics.DisplayLine(GT.Color.Red, 1,
                    x - 1, LastJoyX, x, JoyX); // Draw
                LastJoyX = JoyX;

                int JoyY = (int)(joystick.GetPosition().Y * 15) + 70; // Scale and cast
                displayN18.SimpleGraphics.DisplayLine(GT.Color.Green, 1,
                    x - 1, LastJoyY, x, JoyY); // Draw
                LastJoyY = JoyY;
                if (x++ > 100)
                {
                    x = 2;
                    // Draw a border and clear the inside
                    displayN18.SimpleGraphics.DisplayRectangle(
                        GT.Color.Magenta, 2, GT.Color.Black, 1, 1, 100, 100);
                }
                Thread.Sleep(30);
            }
        }
    }
}

```

13. Memory Storage

Storing files to SD cards and USB memory drives is easily accomplished in .NET Gadgeteer. In fact, you will be using FileStream, the same way files are handled on a computers. Lets read the light sensor voltage level and record it to a file every 50 milliseconds.



This code will wait for the button to be pressed and then it will start recording a Comma Separated Value (CSV) file, which can be opened in excel. Remember to put the SD card in before pressing the button and then press the button again before removing the card. The button has a small LED on it. When this LED is lit, the data is being recorded to the SD card and so it must not be removed.

```
using System;
using System.IO;
using System.Collections;
using System.Threading;
using System.Text;

using GT = Gadgeteer;
using GTM = Gadgeteer.Modules;
using Gadgeteer.Modules.GHIElectronics;

namespace GadgeteerBook
{
    public partial class Program
    {
        Thread RecordingT;
        void ProgramStarted()
        {
            RecordingT = new Thread(Recorder);
            button.ButtonPressed += button_ButtonPressed;
        }

        void button_ButtonPressed(Button sender, Button.ButtonState state)
        {
            if (!button.IsLedOn) // LED is on when we are recording
            {
                button.TurnLedOn();
                RecordingT.Start();
            } else
            {
                button.TurnLedOff();
            }
        }

        void Recorder()
        {
            FileStream f = sdCard.StorageDevice.OpenWrite("Data.csv");
            int counter = 0;
            while (button.IsLedOn)
            {
                // Comma Separated Value, CSV file
                string Line = counter++.ToString() + "," + lightSense.ReadVoltage().ToString()
+ "\r\n";

                byte[] Data = Encoding.UTF8.GetBytes(Line);
                f.Write(Data, 0, Data.Length);
                Thread.Sleep(50);
            }
            f.Close();
        }
    }
}
```

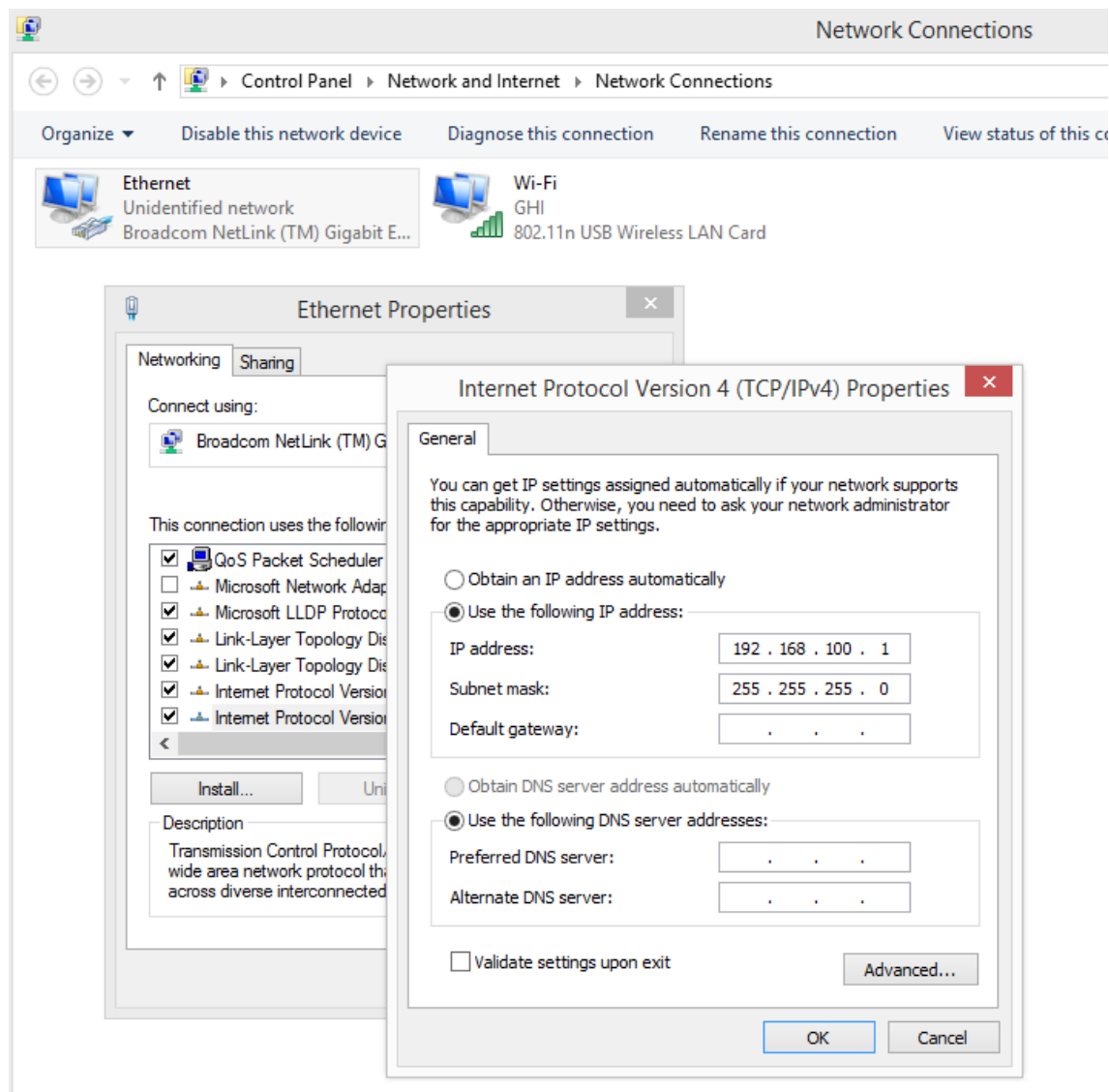
14. Networking

.NET Micro Framework includes Socket support, allowing users to communicate using TCP/UDP. There is also HTTP support for easy web page handling for clients and servers. As for security, SSL is supported on most devices but small devices have XTEA for simple data encryption.

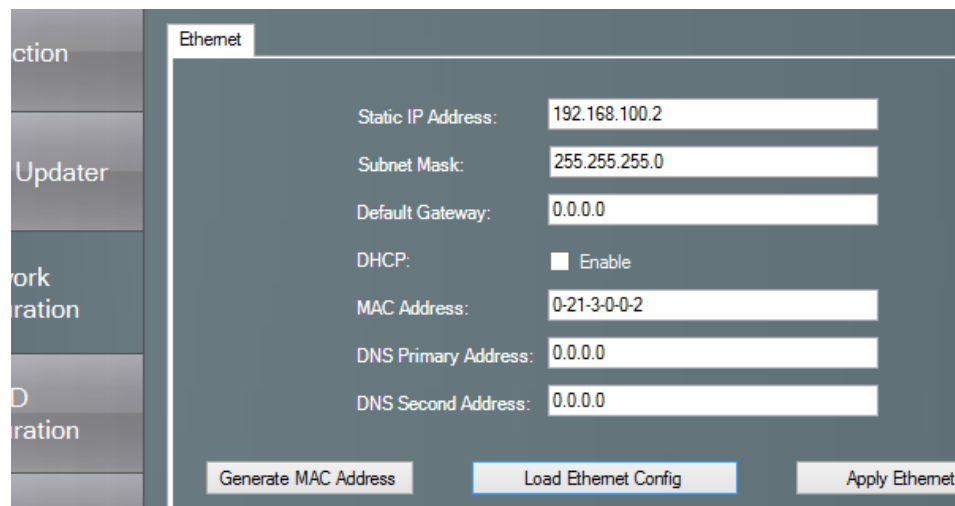
Creating an Internet of Things device with .NET Gadgeteer is very easy as users have all the low level networking they need through .NET Micro Framework but .NET Gadgeteer also extends the support with a user-friendly library for hosting pages.

14.1. Network Settings

We will keep this test as simple as possible. The device will be connected to a computer using an Ethernet cable directly. We will open the network settings on the computer and set the IP address to static with an IP of 192.168.100.1 as shown in the following image.



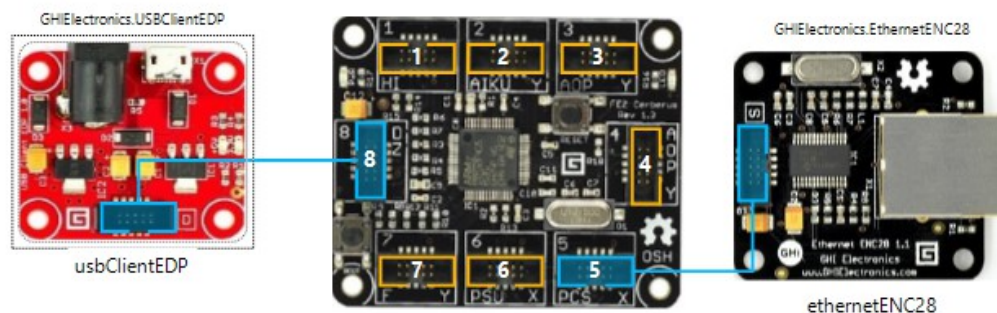
Now set the device with a static IP of 192.168.100.2. This can be done through code, but to keep this example simple, I will use FEZ Config to set the IP address, MFDeploy can also be used as well. Start by clicking the “Load Ethernet Config” button and modify the settings to match the image below then click apply.



You will know if your settings were set properly as the examples provided later will print the IP address on power up.

14.2. Pinging the Device

It's also a good idea to ping the device (network ICMP ping) to make sure everything is working before proceeding. We still need to load a minimal example to the device first. We will start a project with the Ethernet ENC28 Module. This could also be WiFi or any other Networking module.



Now add and run this simple code:

```
using System;  
using System.Collections;  
using System.Threading;
```



```
using System.Text;
using Microsoft.SPOT;
using Gadgeteer.Networking;
using GT = Gadgeteer;
using GTM = Gadgeteer.Modules;
using Gadgeteer.Modules.GHIElectronics;

namespace GadgeteerBook
{
    public partial class Program
    {
        void ProgramStarted()
        {
            ethernetENC28.UseThisNetworkInterface();
            //ethernetENC28.UseStaticIP("")
            ethernetENC28.NetworkUp += ethernetENC28_NetworkUp;
            ethernetENC28.NetworkDown += ethernetENC28_NetworkDown;
            new Thread(RunWebServer).Start();
        }

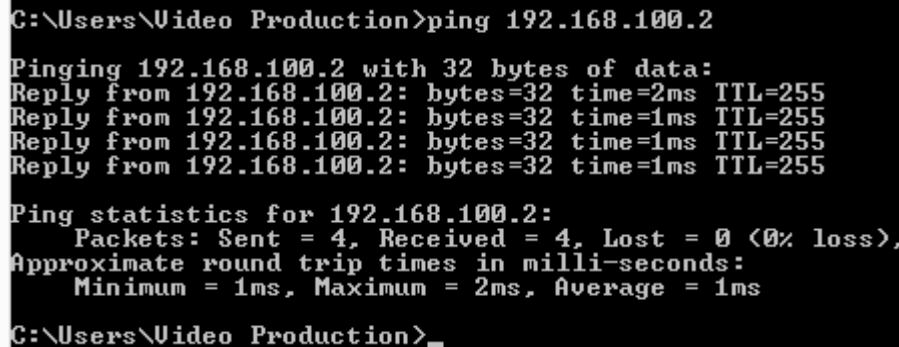
        void ethernetENC28_NetworkDown(GTM.Module.NetworkModule sender,
GTM.Module.NetworkModule.NetworkState state)
        {
            Debug.Print("Network is down!");
        }

        void ethernetENC28_NetworkUp(GTM.Module.NetworkModule sender,
GTM.Module.NetworkModule.NetworkState state)
        {
            Debug.Print("Network is up!");
            Debug.Print("My IP is: " + ethernetENC28.NetworkSettings.IPAddress);
        }

        void RunWebServer()
        {
            // Wait for the network...
            while(ethernetENC28.IsNetworkUp == false)
            {
                Debug.Print("Waiting...");
                Thread.Sleep(1000);
            }
        }
    }
}
```

Open the Command Prompt and ping the device, using **ping 192.168.100.2**

If everything worked as planned then the output will look similar to this image.



```
C:\Users\Video Production>ping 192.168.100.2

Pinging 192.168.100.2 with 32 bytes of data:
Reply from 192.168.100.2: bytes=32 time=2ms TTL=255
Reply from 192.168.100.2: bytes=32 time=1ms TTL=255
Reply from 192.168.100.2: bytes=32 time=1ms TTL=255
Reply from 192.168.100.2: bytes=32 time=1ms TTL=255

Ping statistics for 192.168.100.2:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 1ms, Maximum = 2ms, Average = 1ms

C:\Users\Video Production>
```

14.3. A Known Bug!

There is a known bug in the .NET Gadgeteer code that needs to be fixed before we proceed. The bug is explained at <https://gadgeteer.codeplex.com/workitem/1861>

Once the project is created with a networking module, the references list is automatically populated with the Gadgeteer.WebHost assembly. Remove this assembly manually from the list. Visit the .NET Gadgeteer repository and download the source code files found under:

<https://gadgeteer.codeplex.com/SourceControl/latest#Main/GadgeteerCore/Libraries/Core/WebServer43/>

Include all cs files from there in your project and then open Responder.cs and fix the bug as explained here: <https://gadgeteer.codeplex.com/workitem/1861>

14.4. Network Controlled Devices

This code should be easy to use but you need a basic understanding of HTML, which is the way web pages are constructed. A quick internet search about HTML will help.

```
using System;
using System.Collections;
using System.Threading;
using System.Text;
using Microsoft.SPOT;
using Gadgeteer.Networking;
using GT = Gadgeteer;
```

```
using GTM = Gadgeteer.Modules;
using Gadgeteer.Modules.GHIElectronics;

namespace GadgeteerBook
{
    public partial class Program
    {
        byte[] HTML = Encoding.UTF8.GetBytes(
            "<html><body>" +
            "<h1>Hosted on .NET Gadgeteer</h1>" +
            "<p>Lets scare someone!</p>" +
            "<form action=\"\" method=\"post\">" +
            "<input type=\"submit\" value=\"Toggle LED!\">" +
            "</form>" +
            "</body></html>");

        void ProgramStarted()
        {
            ethernetENC28.UseThisNetworkInterface();
            //ethernetENC28.UseStaticIP("")
            ethernetENC28.NetworkUp += ethernetENC28_NetworkUp;
            ethernetENC28.NetworkDown += ethernetENC28_NetworkDown;
            new Thread(RunWebServer).Start();
        }

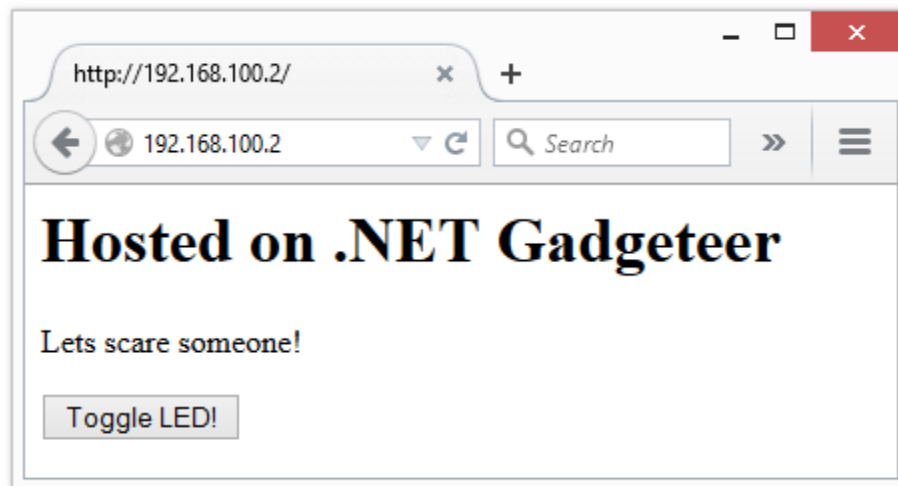
        void ethernetENC28_NetworkDown(GTM.Module.NetworkModule sender,
            GTM.Module.NetworkModule.NetworkState state)
        {
            Debug.Print("Network is down!");
        }

        void ethernetENC28_NetworkUp(GTM.Module.NetworkModule sender,
            GTM.Module.NetworkModule.NetworkState state)
        {
            Debug.Print("Network is up!");
            Debug.Print("My IP is: " + ethernetENC28.NetworkSettings.IPAddress);
        }

        void RunWebServer()
        {
            // Wait for the network...
            while(ethernetENC28.IsNetworkUp == false)
            {
                Debug.Print("Waiting...");
                Thread.Sleep(1000);
            }
            // Start the server
            WebServer.StartLocalServer(ethernetENC28.NetworkSettings.IPAddress, 80);
            WebServer.DefaultEvent.WebEventReceived += DefaultEvent_WebEventReceived;
            while (true)
            {
                Thread.Sleep(1000);
            }
        }
    }
}
```

```
    }  
  }  
  bool LedState = false;  
  void DefaultEvent_WebEventReceived(string path, WebServer.HttpMethod method, Responder  
  responder)  
  {  
    // We always send the same page back  
    responder.Respond(HTML, "text/html; charset=utf-8");  
  
    // If a button was clicked  
    if (method == WebServer.HttpMethod.POST)  
    {  
      LedState = !LedState; // Toggle state  
      Mainboard.SetDebugLED(LedState);  
    }  
  }  
}
```

In this example, a .NET Gadgeteer system is connected to a computer directly with IP addresses set as explained earlier. Open any web browser and type in the IP address of the device. This should show the page we are hosting:



The code above handles any page requested as a default, meaning accessing page <http://192.168.100.2/about> will result in the same page. Let us add a WebEvent that handles the about page with a different result. Test the code below by visiting: <http://192.168.100.2/about>

```
using System;
using System.Collections;
using System.Threading;
using System.Text;
using Microsoft.SPOT;
using Gadgeteer.Networking;
using GT = Gadgeteer;
using GTM = Gadgeteer.Modules;
using Gadgeteer.Modules.GHIElectronics;

namespace GadgeteerBook
{
    public partial class Program
    {
        byte[] HTML = Encoding.UTF8.GetBytes(
            "<html><body>" +
            "<h1>Hosted on .NET Gadgeteer</h1>" +
            "<p>Lets scare someone!</p>" +
            "<form action=\"\" method=\"post\">" +
            "<input type=\"submit\" value=\"Toggle LED!\">" +
            "</form>" +
            "</body></html>");

        void ProgramStarted()
        {
            ethernetENC28.UseThisNetworkInterface();
            //ethernetENC28.UseStaticIP("")
            ethernetENC28.NetworkUp += ethernetENC28_NetworkUp;
            ethernetENC28.NetworkDown += ethernetENC28_NetworkDown;
            new Thread(RunWebServer).Start();
        }

        void ethernetENC28_NetworkDown(GTM.Module.NetworkModule sender,
            GTM.Module.NetworkModule.NetworkState state)
        {
            Debug.Print("Network is down!");
        }

        void ethernetENC28_NetworkUp(GTM.Module.NetworkModule sender,
            GTM.Module.NetworkModule.NetworkState state)
        {
            Debug.Print("Network is up!");
            Debug.Print("My IP is: " + ethernetENC28.NetworkSettings.IPAddress);
        }

        void RunWebServer()
        {
            // Wait for the network...
            while(ethernetENC28.IsNetworkUp == false)
            {
            }
        }
    }
}
```

```
        Debug.Print("Waiting...");
        Thread.Sleep(1000);
    }
    // Start the server
    WebServer.StartLocalServer(ethernetENC28.NetworkSettings.IPAddress, 80);
    WebEvent webEvent = WebServer.SetupWebEvent("about");
    webEvent.WebEventReceived += webEvent_WebEventReceived;
    WebServer.DefaultEvent.WebEventReceived += DefaultEvent_WebEventReceived;
    while (true)
    {
        Thread.Sleep(1000);
    }
}
bool LedState = false;
void DefaultEvent_WebEventReceived(string path, WebServer.HttpMethod method, Responder
responder)
{
    // We always send the same page back
    responder.Respond(HTML, "text/html;charset=utf-8");

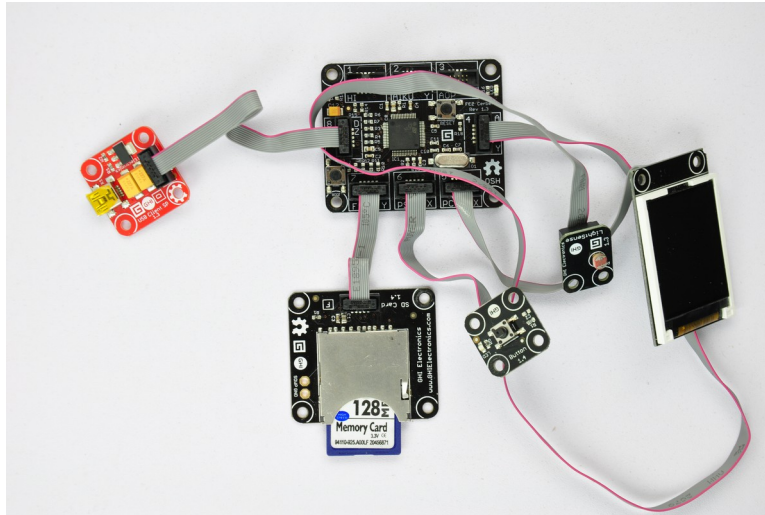
    // If a button was clicked
    if (method == WebServer.HttpMethod.POST)
    {
        LedState = !LedState; // Toggle state
        Mainboard.SetDebugLED(LedState);
    }
}

void webEvent_WebEventReceived(string path, WebServer.HttpMethod method, Responder
responder)
{
    byte[] AboutHTML = Encoding.UTF8.GetBytes(
        "<html><body>" +
        "<p>.NET Gadgeteer is just amazing, and Freakin' Easy to use!</p>" +
        "</body></html>");

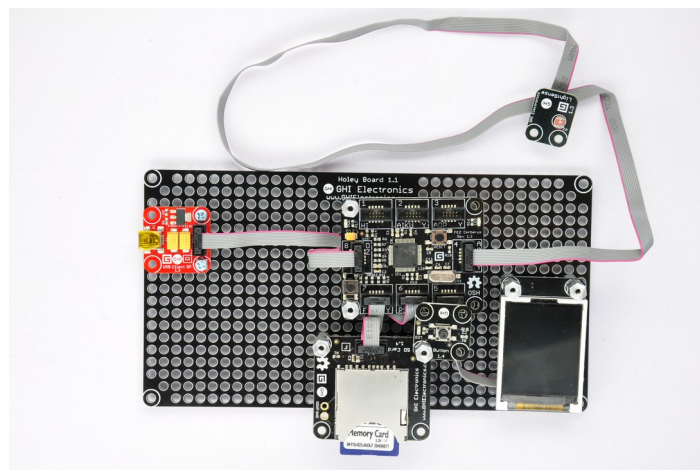
    responder.Respond(AboutHTML, "text/html;charset=utf-8");
}
}
```

15. Keeping it Tidy!

As your projects grows, keeping the modules and cables organized can be a problem. Here is a picture of a project; with just five modules, things are looking twisted and tangled.



The alternative is to mount the modules on the “Holey Board” using some stand-offs and screws.



16. Gadgeteering the non-Gadgeteer!

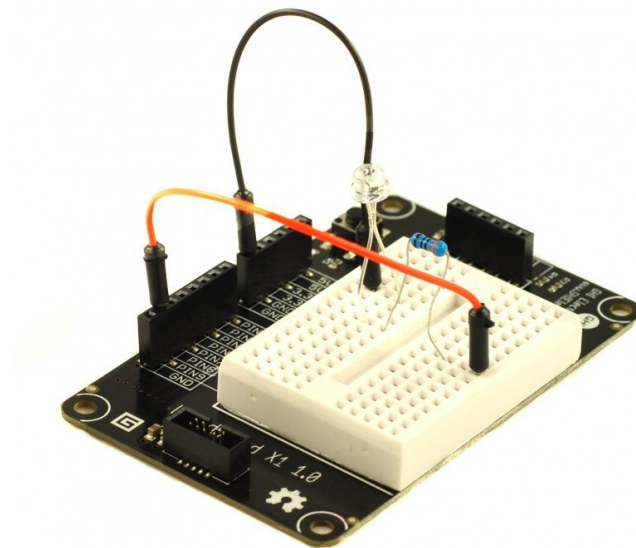
There are cases when a specific sensor is required to be used and there is no .NET Gadgeteer module for it. Fear not, almost anything can be wired to a .NET Gadgeteer system. This socket map pin-out shows what kind of signals are available on sockets. For example, the A socket has 3 analog pins on pins 3, 4 and 5, with a GPIO on pin 6.

Socket	Pin1	Pin2	Pin3	Pin4	Pin5	Pin6	Pin7	Pin8	Pin9	Pin10
<u>A</u>	+3.3V	+5V	AIN (G!)	AIN (G)	AIN	GPIO	[UN]	[UN]	[UN]	GND
<u>B</u>	+3.3V	+5V	LCD B0	LCD B1	LCD B2	LCD B3	LCD B4	LCD ENABLE	LCD CLK	GND
<u>C</u>	+3.3V	+5V	GPIO!	CAN TD (G)	CAN RD (G)	GPIO	[UN]	[UN]	[UN]	GND
<u>D</u>	+3.3V	+5V	GPIO!	D-	D+	GPIO	GPIO	[UN]	[UN]	GND
<u>E</u>	+3.3V	+5V	[UN]	LED1 (OPT)	LED2 (OPT)	TX D-	TX D+	RX D-	RX D+	GND
<u>F</u>	+3.3V	+5V	GPIO!	DAT0	DAT1	CMD	DAT2	DAT3	CLK	GND
<u>G</u>	+3.3V	+5V	LCD G0	LCD G1	LCD G2	LCD G3	LCD G4	LCD G5	LCD BACKLIGHT	GND
<u>H</u>	+3.3V	+5V	GPIO!	D-	D+	[UN]	[UN]	[UN]	[UN]	GND
<u>I</u>	+3.3V	+5V	GPIO!	[UN]	[UN]	GPIO	[UN]	SDA	SCL	GND
<u>K</u>	+3.3V	+5V	GPIO!	TX (G)	RX (G)	RTS	CTS	[UN]	[UN]	GND
<u>Q</u>	+3.3V	+5V	GPIO!	GPIO	AOUT	[UN]	[UN]	[UN]	[UN]	GND
<u>P</u>	+3.3V	+5V	GPIO!	[UN]	[UN]	GPIO	PWM (G)	PWM (G)	PWM	GND
<u>R</u>	+3.3V	+5V	LCD R0	LCD R1	LCD R2	LCD R3	LCD R4	LCD VSYNC	LCD HSYNC	GND
<u>S</u>	+3.3V	+5V	GPIO!	GPIO	GPIO	CS	MOSI	MISO	SCK	GND
<u>I</u>	+3.3V	+5V	[UN]	YU	XL	YD	XR	[UN]	[UN]	GND
<u>U</u>	+3.3V	+5V	GPIO!	TX (G)	RX (G)	GPIO	[UN]	[UN]	[UN]	GND
<u>X</u>	+3.3V	+5V	GPIO!	GPIO	GPIO	[UN]	[UN]	[UN]	[UN]	GND
<u>Y</u>	+3.3V	+5V	GPIO!	GPIO	GPIO	GPIO	GPIO	GPIO	GPIO	GND
<u>Z</u>	+3.3V	+5V	[MS]	[MS]	[MS]	[MS]	[MS]	[MS]	[MS]	GND
<u>.</u>	+3.3V	+5V	GPIO!	GPIO	GPIO	[MS]	[MS]	[MS]	[MS]	GND

This is the Legend for the pin-out:

Symbol	Description
AIN	Analog input pin.
GPIO	A general-purpose digital input/output pin, operating at 3.3 volts.
(G)	In addition to another functionality, a pin that is also usable as a GPIO.
[UN]	Modules must not connect to this pin if using this socket type. Mainboards can support multiple socket types on one socket, as long as individual pin functionalities overlap in a compatible manner, so that a pin from one socket type can overlap with a [UN] pin of another.
!	Interrupt-capable and software pull-up capable GPIO (the pull-up is switchable and in the range of 10,000 to 100,000 ohms).
+3.3V	Connection to the +3.3 V power net.
+5V	Connection to the +5 V power net.
GND	Connection the power ground net.

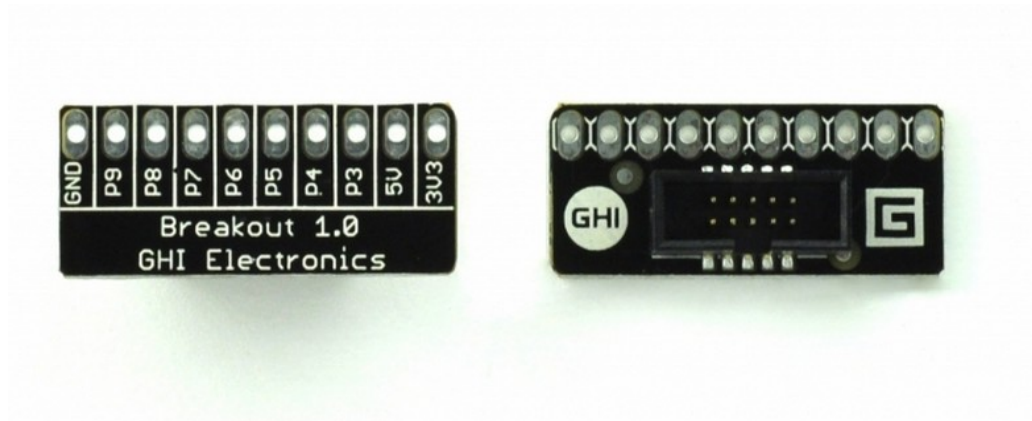
If you are familiar with breadboards, using the breadboard module is probably the best route.



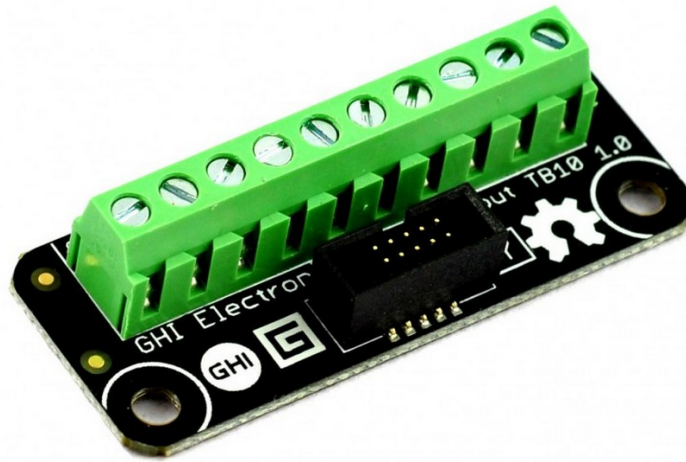
Assuming a sensor has an analog output that is suited for 3.3V levels found on Gadgeteer mainboards, the sensor can be directly connected to any socket type A, to pin 3, 4 or 5. If the

signal needs conditioning, perhaps the output from the sensor is 0V to 0.1V then the signal conditioning circuitry can be prototyped right on the breadboard.

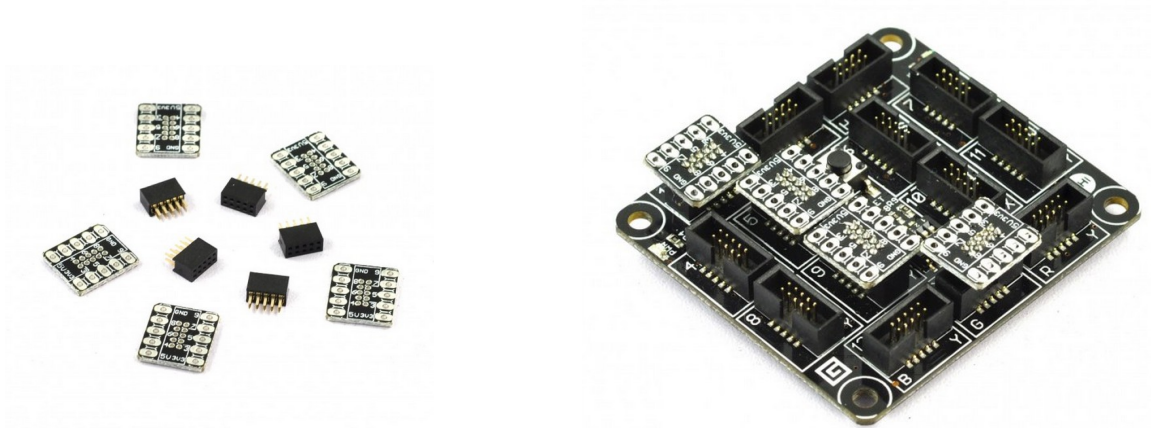
There are other modules that can be used as well. Take the breakout module for example. This module comes in two styles. The first one is small, with holes for soldering wires or headers.



The second one has a terminal block. A small screwdriver can be used to tighten the screws that hold the wires in place.



Another very tiny module is the G-Plug. This module plugs right on the socket and is exactly the same size as the box surrounding sockets on the mainboard. This allows multiple sockets to be plugged in without overlapping.



16.1. Accessing the Pins

There are two ways to access the pins, using .NET Micro Framework directly or through the .NET Gadgeteer helpers. Going to NETMF directly is desired if the design will be moved to a non-Gadgeteer production. To determine which processor pin is connected to what pin on what socket, the schematic will be needed. This cuts out the need to include the .NET Gadgeteer core but then the code will hard-coded to a processor pin on a certain board on a certain processor. Moving the pin will require going back to the schematics.

The other way to keep the code dynamic but .NET Gadgeteer software will be required. If this is unclear, I recommend sticking to .NET Gadgeteer projects. This is an example on how to obtain the processor pin on socket number 2, pin number 3.

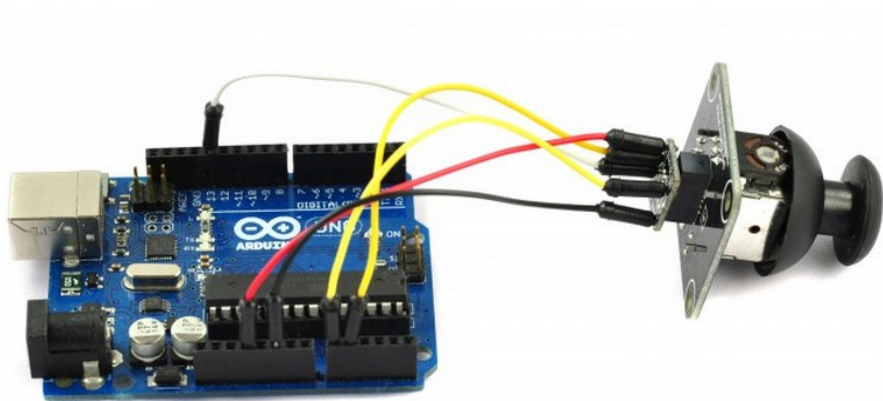
```
Cpu.Pin pin = GT.Socket.GetSocket(2, true, null, null).CpuPins[3];
```

This document has further information:

<https://www.ghielectronics.com/docs/144/plain-netmf-on-gadgeteer>

16.2. Not using Mainboards

The .NET Gadgeteer modules can also be used with non-Gadgeteer processor boards. The website details the pin-out of each module and provides the schematics. This example shows the Joystick Module connected to an Arduino board.



17. Visual Basic

Developers that come from embedded development will usually prefer C# since C is the main language used. However, Visual Basic is also very popular and simpler to work with. Both languages have grown so much that they are as capable. A person will decide on one vs another only based on past experience, not capabilities. If you are not familiar with either, C# is recommended. When stepping outside .NET Micro Framework, to load a complex routing through RLP, C is the language to be used. Since C and C# are from the same family, learning C# will be the better path down the road.

Both languages are so close in capabilities that there are websites that can do auto conversion for you. You can also do the conversion manually if preferred. To test this, the converter on Telerik website was used to convert this simple code.

Here is the VB code followed by the C# code. Do you see the similarities?

```
Imports GT = Gadgeteer
Imports GTM = Gadgeteer.Modules

Namespace HelloLED
    Partial Public Class Program

        Private Sub ProgramStarted()
            Debug.Print("Hello PC")' Print something
            Mainboard.SetDebugLED(True)' Turn the LED on
        End Sub
    End Class
End Namespace
```

```
using System;
using GT = Gadgeteer;
using GTM = Gadgeteer.Modules;

namespace HelloLED
{
    public partial class Program
    {
        void ProgramStarted()
        {
            Debug.Print("Hello PC");// Print something
            Mainboard.SetDebugLED(true);// Turn the LED on
        }
    }
}
```

18. Prototype to Production

While .NET Gadgeteer makes prototyping extremely fast and convenient, it may not be suitable for mid to high run volumes. Bringing all needed components into one circuit board makes the final board smaller and brings the cost down.

18.1. The Hardware

GHI Electronics offers System on Module (SoM) or System on Chip (SoC) solutions that run .NET Micro Framework. Those can be easily leveraged on the final product, where GHI Electronics maintain the .NET Micro Framework with free updates.

Minimal hardware design experience is still required to design circuit boards. The custom design can be assigned to a third party or to GHI Electronics. The cost is typically reasonable as the most complex part of the design is done, thanks to GHI Electronics' SoM and SoC.

18.2. The Software

.NET Gadgeteer is a helper software that runs on top of .NET Micro Framework. It is completely up to the designer to continue on using .NET Gadgeteer or go directly to .NET Micro framework. A mix of both maybe desired in some cases. The best part is that the code can be moved from the .NET Gadgeteer prototype to production board with no or minimal changes.

18.3. .NET Gadgeteer in Production

On designs where time to market is absolutely critical or volumes are low, shipping a product with .NET Gadgeteer hardware is desired. This is where 3D printing or laser cut acrylic can be utilized to its fullest.

