Getting started with Mountaineer

.NET Gadgeteer compatible platform

Welcome to the Mountaineer Platform!

You've made a good choice! The Mountaineer mainboard you just received is a great way of quickly and efficiently building small intelligent devices using .NET Gadgeteer, the .NET Micro Framework (NETMF), and the Visual Studio development environment.

The Mountaineer mainboards are designed to work with a minimal number of components. While other systems need separate modules such as power supply modules, both Mountaineer mainboards allow you to get started straight away, just by connecting the mainboard to a PC through the standard micro-USB cable that comes with the board.

Your very first Gadgeteer program can therefore run with a minimal amount of cable clutter. Using the Ethernet mainboard even allows you to connect to the Internet without the need of any external module. Don't worry, Gadgeteer-compatible third-party modules will work with these mainboards as well. But for now, we'll focus on the on-board capabilities of the boards.

Step 1: install the development environment and the drivers

You'll find the links to the latest development software as well as all the installation instructions on the Mountaineer website.

Step 2: connect the hardware

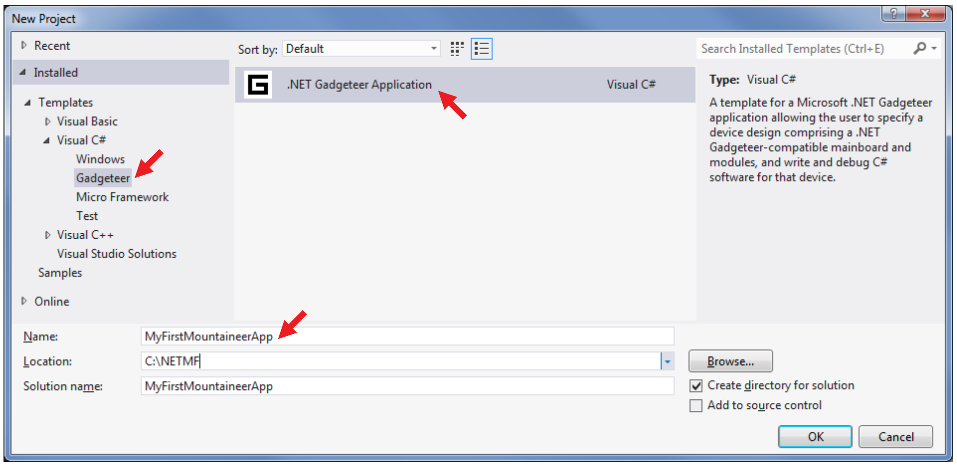
It can't be easier: plug the provided micro-USB cable into a free USB port of your PC on one side, and into the Mountaineer mainboard on the other side. That's it, your Mountaineer mainboard is now ready for programming!

Step 3: prepare your first application with a Mountaineer mainboard

Launch Microsoft Visual Studio or Microsoft Visual Studio Express (hereafter called VS) and follow these steps:

1) Click File >> New project...
2) Under Installed >> Templates, expand the Visual C# tab and click on Gadgeteer
3) Select .NET Gadgeteer Application
4) Give a name to your project (for example „MyFirstMountaineerApp") and click OK

Note: the screenshots and menus are shown here for Visual Studio 2012 Express and can slightly differ for other versions.



.NET Gadgeteer Application — Visual C#

Type: Visual C#

A template for a Microsoft .NET Gadgeteer application allowing the user to specify a device design comprising a .NET Gadgeteer-compatible mainboard and modules, and write and debug C# software for that device.

Name: MyFirstMountaineerApp

Location: C:\NETMF

Solution name: MyFirstMountaineerApp

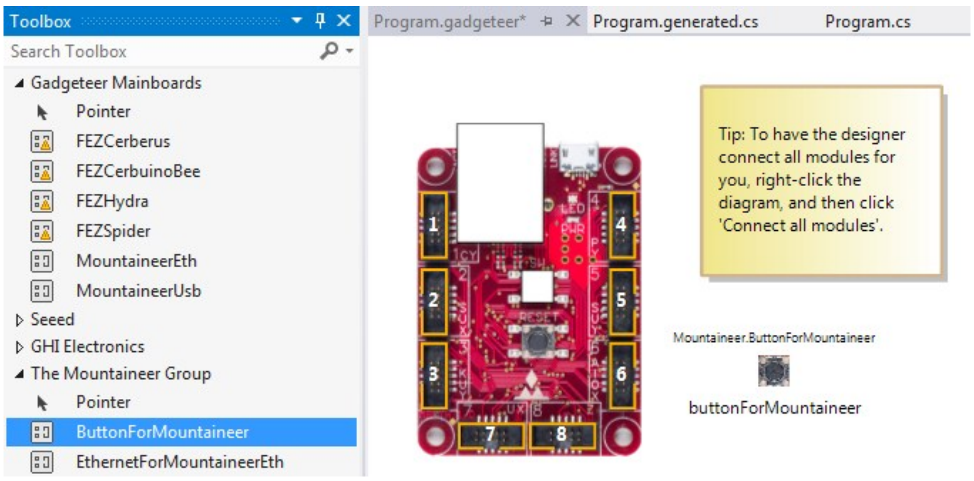
☑ Create directory for solution

☐ Add to source control

In the application wizard, you'll then be asked to choose a mainboard. Let's select for example the Ethernet Mainboard. In this window, you can also choose which framework version you'll use. Once done, click "Create".

VS now opens the .NET Gadgeteer Designer, allowing you to virtually connect the modules you need to the mainboard. For this first application, you'll just use the built-in user button and the Debug LED:

1) Check that the Toolbox is visible. If not, click View >> Toolbox to display it. This toolbox gives you access to all installed Gadgeteer mainboards and modules, classified by manufacturer.
2) Your mainboard should already be placed in the designer. If not, drag & drop it from the toolbox onto the Designer main screen.
3) Drag & drop the module "ButtonForMountaineer" (in the register "The Mountaineer Group") from the toolbox onto the Designer main screen, next to the mainboard. You should now see something like this:



The white squares on the mainboard picture mean that the mainboard is by default not initialized with any built-in module. That's why you need to add the button as described above.

There's no need of connecting the button together with the mainboard because this module is already embedded in the hardware of the mainboard. Adding this built-in module on the Designer simply tells VS that the mainboard must be initialized to support it.

Now, after saving this design (CTRL+S), you've got your mainboard automatically initialized. In the Solution Explorer, check out the file "Program.generated.cs", you'll see the code the Designer wrote for you.

Step 4: write the code

Go back to "Program.cs". This file is where you'll have to write your code. You'll see here how to use some basic principles of Gadgeteer: the timers, the event handlers, etc. The program you're going to write will make the Debug LED blink a couple of times when the button is pressed. The blinking will be handled by a timer, whereas the button events will be managed by an event handler.

Your program will run from the method `ProgramStarted`, already implemented in the template.

Let's first define some global variables for the class `Program`:

```
public partial class Program
{
    /// <summary>
    /// Defines how many times the LED must blink when pressing the button
    /// </summary>
    private const int BlinkTimes = 10;

    /// <summary>
    /// Defines the blink frequency. This time (milliseconds) defines the ON and OFF time intervals of LED
    /// </summary>
```

```
        private const int BlinkTimeMs = 100;

        /// <summary>
        /// Used to set the state of the LED: true = on / false = off
        /// </summary>
        private bool ledState = true;

        /// <summary>
        /// Used to count how many times the LED already blinked
        /// </summary>
        private int loopIndex = 0;

        /// <summary>
        /// Gadgeteer timer for blinking
        /// </summary>
        private GT.Timer blinkTimer;
```

Note: the XML comments (beginning with a triple slash) are not mandatory, but it's a good practice to use them for documentation purposes.

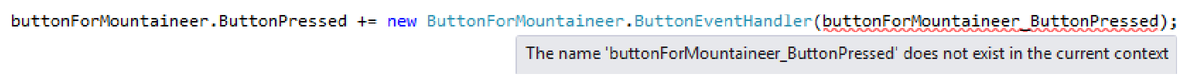
Next, initialize the Gadgeteer timer (just follow the example of the template) when entering the method `ProgramStarted`. Then initialize the event handlers to manage both button events "pressed" and "released":

```
            // Use Debug.Print to show messages in Visual Studio's "Output" window during debugging.
            Debug.Print("Program Started");

            blinkTimer = new GT.Timer(BlinkTimeMs);
            blinkTimer.Tick += new GT.Timer.TickEventHandler(blinkTimer_Tick);
            Debug.Print("Timer initialized");

            buttonForMountaineer.ButtonPressed += new ButtonForMountaineer.ButtonEventHandler(buttonForMountaineer_ButtonPressed);
            buttonForMountaineer.ButtonReleased += new ButtonForMountaineer.ButtonEventHandler(buttonForMountaineer_ButtonReleased);
            Debug.Print("Event handler linked");
        }
```

If you type the code (instead of copying and pasting it from here), you'll enjoy the automated code generation for the event handler when typing "+=" and pressing the "Tab" key twice. This will also generate the methods showed later in this document. If you copy the code from here, VS will display this error message when pointing the cursor on the event handler method:



```
buttonForMountaineer.ButtonPressed += new ButtonForMountaineer.ButtonEventHandler(buttonForMountaineer_ButtonPressed);
                                             The name 'buttonForMountaineer_ButtonPressed' does not exist in the current context
```

Don't worry, this error is normal because the method `buttonForMountaineer_ButtonPressed` doesn't exist yet. This will be fixed soon.

Now that the "pressed" and "released" button events can be detected, you have to describe which action you want to undertake when this happens. Let's decide the following:

- When pressing the button, start the blink timer for the LED
- When releasing it, do nothing but display a debug message (just to ensure it works)

Here are the two methods to handle these events (put them outside of `ProgramStarted`, still in the class `Program`):

```
        private void buttonForMountaineer_ButtonPressed(ButtonForMountaineer sender, ButtonForMountaineer.ButtonState value)
        {
            Debug.Print("Button pressed!");
            blinkTimer.Start();
        }

        private void buttonForMountaineer_ButtonReleased(ButtonForMountaineer sender, ButtonForMountaineer.ButtonState value)
        {
            Debug.Print("Button released!");
        }
```

Lastly, let's implement the method for blinking the LED. Since this method is handled by the timer, it will be periodically repeated. So basically, you just have to invert the LED state and count up to the defined blink times, for example 10 times. When this is reached, stop the timer and reset the counter:

```
        private void blinkTimer_Tick(GT.Timer blinkTimer)
        {
            if (loopIndex != 2 * BlinkTimes)       // 2 * BlinkTimes because one blink needs two ticks (ON and OFF)
            {
                if (ledState)
                {
                    Mainboard.SetDebugLED(true);
                }
                else
                {
                    Mainboard.SetDebugLED(false);
                }
                ledState = !ledState;
                loopIndex++;
            }
            else
            {
                blinkTimer.Stop();
                loopIndex = 0;
            }
        }
    }
}
```

Your button event handler is still ready to trigger, so the next loop will start when pressing the button again.

Step 5: build and deploy to the hardware

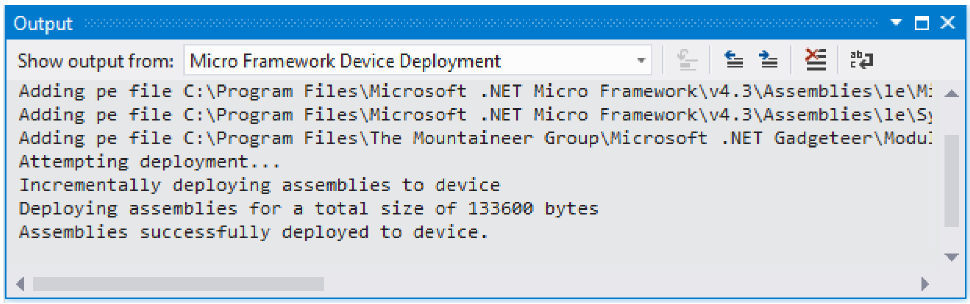
Now that your code is ready, it can be built and deployed to the device's Flash memory. Let's first check the configuration for the deployment:

1) Click Project >> MyFirstMountaineerApp Properties
2) Select the ".NET Micro Framework" register
3) Check that the following is selected: "Transport: USB" and "Device: MountaineerEth" (or "MountaineerUsb "depending on the board you use).

Now you can build your code:

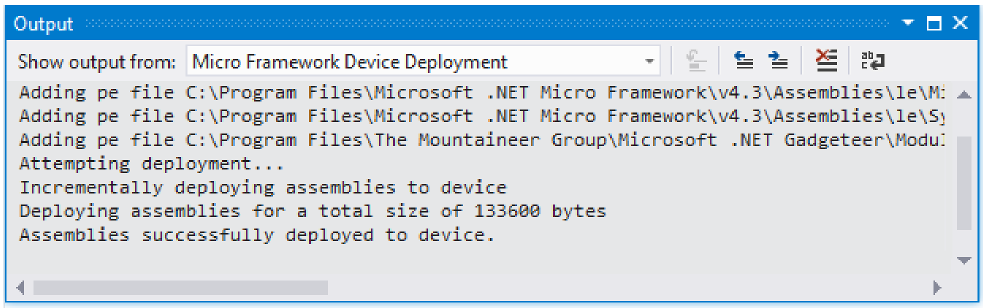
1) Click Build >> Build Solution (F7 key). You should see "Build succeeded" on the status bar. Otherwise, an error list will appear on the screen, telling you what's wrong in the code.
2) Click Debug >> Start Debugging (F5 key). The program will now be loaded ("deployed") onto the hardware.

VS tells you what happens during the deployment in the "Output" window. If you don't see this window, click View >> Output to display it.



```
Adding pe file C:\Program Files\Microsoft .NET Micro Framework\v4.3\Assemblies\le\M
Adding pe file C:\Program Files\Microsoft .NET Micro Framework\v4.3\Assemblies\le\S
Adding pe file C:\Program Files\The Mountaineer Group\Microsoft .NET Gadgeteer\Modu
Attempting deployment...
Incrementally deploying assemblies to device
Deploying assemblies for a total size of 133600 bytes
Assemblies successfully deployed to device.
```

After waiting the end of the deployment, your code should be directly working on the board. Press the button labelled “SW” on your Mountaineer mainboard to try it out! You should see the debug messages in the “Output” window:



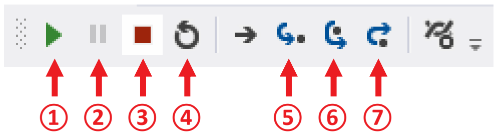
```
Adding pe file C:\Program Files\Microsoft .NET Micro Framework\v4.3\Assemblies\le\M:
Adding pe file C:\Program Files\Microsoft .NET Micro Framework\v4.3\Assemblies\le\Sy
Adding pe file C:\Program Files\The Mountaineer Group\Microsoft .NET Gadgeteer\Modul
Attempting deployment...
Incrementally deploying assemblies to device
Deploying assemblies for a total size of 133600 bytes
Assemblies successfully deployed to device.
```

If you see these messages, the LED should blink as expected. Congratulations, you just wrote your first Mountaineer application successfully!

Step 6: debugging

Does your program work as expected? Do you experience some problems during its execution? Doesn’t it work at all? No problem, VS gives you the opportunity to debug your code by going step by step or setting breakpoints in the code.

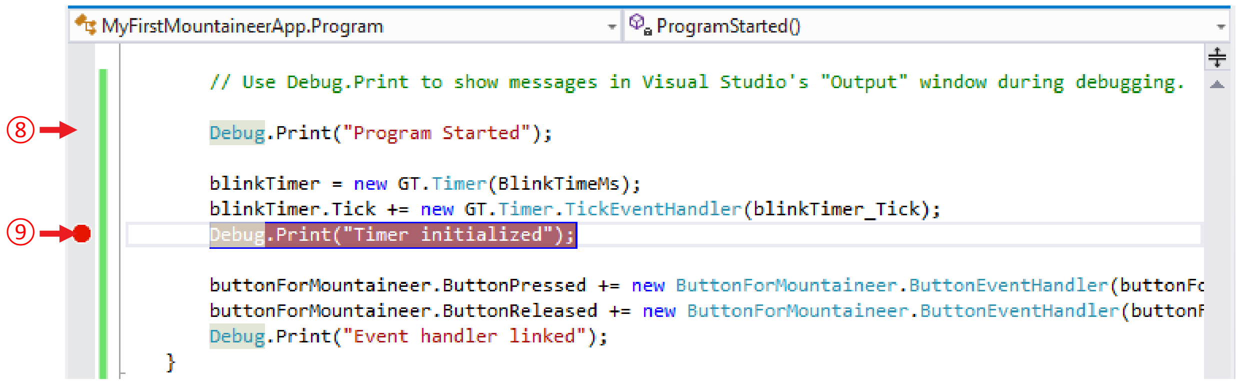
Is your debug toolbar active? It should look like this:



If not, you’ll find it here: click View >> Toolbars >> Debug.

These tools allow you executing the program or restarting after a breakpoint (①), stopping the execution of the program (②), stopping the debug session (③), restarting the debug session and the board (④), going one step forward (⑤), stepping over a statement (⑥, for example a for loop), or stepping out of a statement (⑦).

The program execution will stop when encountering a breakpoint. The breakpoints are very useful if you want to execute your code up to a certain line. In this case, set a breakpoint where you want the program to stop simply by clicking on the light grey margin on the left of the code (⑧):



```
    // Use Debug.Print to show messages in Visual Studio's "Output" window during debugging.

    Debug.Print("Program Started");

    blinkTimer = new GT.Timer(BlinkTimeMs);
    blinkTimer.Tick += new GT.Timer.TickEventHandler(blinkTimer_Tick);
    Debug.Print("Timer initialized");

    buttonForMountaineer.ButtonPressed += new ButtonForMountaineer.ButtonEventHandler(buttonFc
    buttonForMountaineer.ButtonReleased += new ButtonForMountaineer.ButtonEventHandler(buttonF
    Debug.Print("Event handler linked");
}
```

Just click once on any breakpoint (⑨) to remove it.

Step 7: start playing!

Now it's your turn! Add external modules to your mainboard, develop, deploy, debug and… have fun with Mountaineer!

Useful links

Mountaineer

- http://www.mountaineer.org
- http://www.mountaineer-boards.com

Gadgeteer

- http://www.netmf.com/gadgeteer
- http://gadgeteer.codeplex.com

Licensing