

# Getting Started with the FEZ Spider Kit for Microsoft .NET Gadgeteer

Welcome to the FEZ Spider Kit from [GHI Electronics](#). This kit is compatible with [Microsoft .NET Gadgeteer](#). The FEZ Spider Kit enables you to quickly prototype and test a wide variety of functionality for embedded devices. This guide introduces you to the basic hardware components of the FEZ Spider Kit, and shows you how to create your first .NET Gadgeteer application.

.NET Gadgeteer development requires that you have Microsoft Visual Studio installed on your computer. You can use either of the following Visual Studio packages:

- [Microsoft Visual Studio 2010](#) - This is the full featured Visual Studio application development suite with support for multiple programming languages.
- [Microsoft Visual C# 2010 Express](#) - A free alternative, Visual C# 2010 Express provides lightweight, easy-to-learn and easy-to-use tools for creating applications.

This guide is organized into the following sections:

- Building your First .NET Gadgeteer Device
- Creating Your First .NET Gadgeteer Application
- Deploying Your .NET Gadgeteer Application
- Troubleshooting
- 

## ***Building your First .NET Gadgeteer Device***

The FEZ Spider Kit consists of components, which are called modules, and cables that you can use to create various types of functionality in your device. To create your first .NET Gadgeteer device, you will need the following parts:

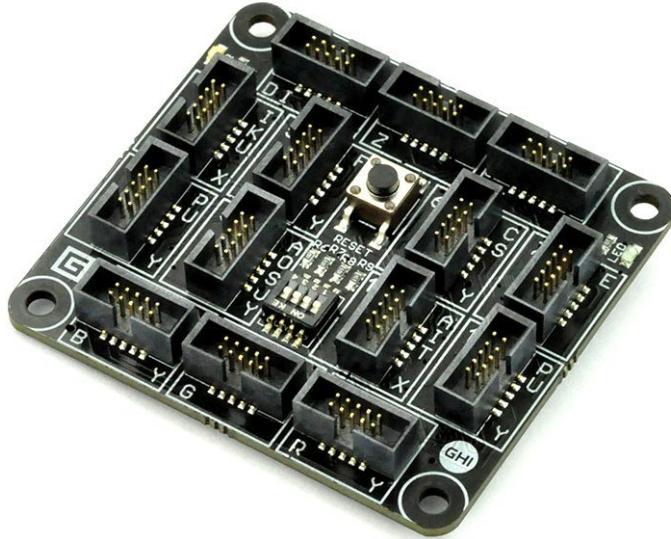
- FEZ Spider Mainboard
- A **Red USB Client Dual Power (USBClientDP) module**
- A **Button** module
- A **Camera** module
- A **Display\_T35** module
- Module connector cables

## **The FEZ Spider Mainboard**

The FEZ Spider Mainboard is the most important part of the GHI Electronics FEZ Spider Kit. The FEZ Spider Mainboard includes a processor and memory, as well as 14 sockets. The sockets are outlined by a white box that surrounds the socket number (**1** through **14**) and groups the socket number with a set of letters that indicate which modules can be connected to the socket.

.NET Gadgeteer-compatible hardware modules connected to the FEZ Spider mainboard by these connectors allow you to extend the FEZ Spider mainboard with communication, user interaction, sensing, and actuation capabilities.

The FEZ Spider mainboard includes a Reset button to reboot the system. There is also a small LED (labeled D1) which lights up whenever the FEZ Spider has power.



## The USB Client Dual Power Device Module

The **USB Client Dual Power (USBClientDP)** module (colored red) enables you to connect the FEZ Spider Mainboard to your computer for programming and debugging. The dual-powered module is itself powered either by a USB port on a computer or by a 7 -30 volt DC power source. A **USBClientDP** module supplies power to the FEZ Spider and to any other modules that are connected to it. You can plug in both power sources of the **USBClientDP** module to program and to power at the same time.



### Warning

Never connect more than one red module to the FEZ Spider Mainboard at the same time. This will damage the hardware.

The **USBClientDP** module has a black socket, identical to the sockets on the FEZ Spider Mainboard. Next to the connector, there is a letter **D**. **This means that this particular module can only be connected to a socket labelled D on the mainboard.**

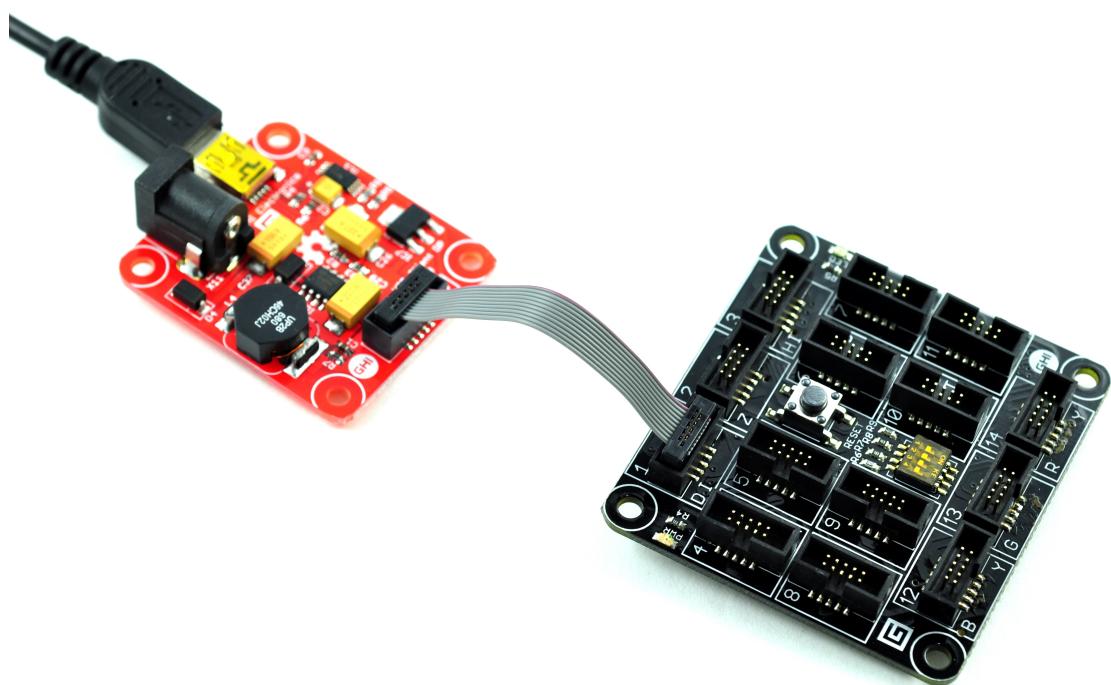
In a similar way, all .NET Gadgeteer-compatible modules have letters next to their sockets that identify which mainboard sockets they can be connected to. Many modules are labeled with multiple letters. This means that they can be connected to **any** of the labeled sockets.

## The Module Connector Cable

Your hardware kit includes many module connector cables of different lengths. Apart from the length, these cables are all identical and can be used interchangeably to connect modules to the FEZ Spider Mainboard. Note that all sockets have a notch and the cable headers have a protrusion that fits into this notch, so the cables can only be inserted one way.



Connect the **red** USB Device module to socket number 1 on the FEZ Spider Mainboard, which is the only socket that has the letter D. Then, connect the small end of the mini USB cable provided with the kit to the USBClientDP module. However, **do not connect the other end to your computer yet.**



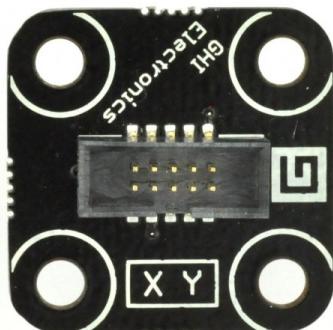
### Warning

When plugging or unplugging any module into a FEZ Spider socket, always make sure that power is not connected, by unplugging either end of the mini USB cable. The mini USB cable supplies power to the FEZ Spider; if you plug or unplug a module on the FEZ Spider while it is powered, the hardware could be damaged.

After ensuring that the FEZ Spider mainboard is not powered, continue by getting a **Button** module from your hardware kit.



Turn the **Button** module over. Next to the connector are the letters **X Y**. This means that a **Button** module can be connected to one of the sockets labeled **X** or **Y** on a mainboard.



Get a **Camera** module from your hardware kit. Turn the **Camera** module over. This module has a single connector labeled **H**. Only socket **3** on the FEZ Spider mainboard supports modules labeled with the letter **H**. Plug one end of a connector cable to the socket on the **Camera** module and the other end to the FEZ Spider on socket number **3**, also labeled **HI**.

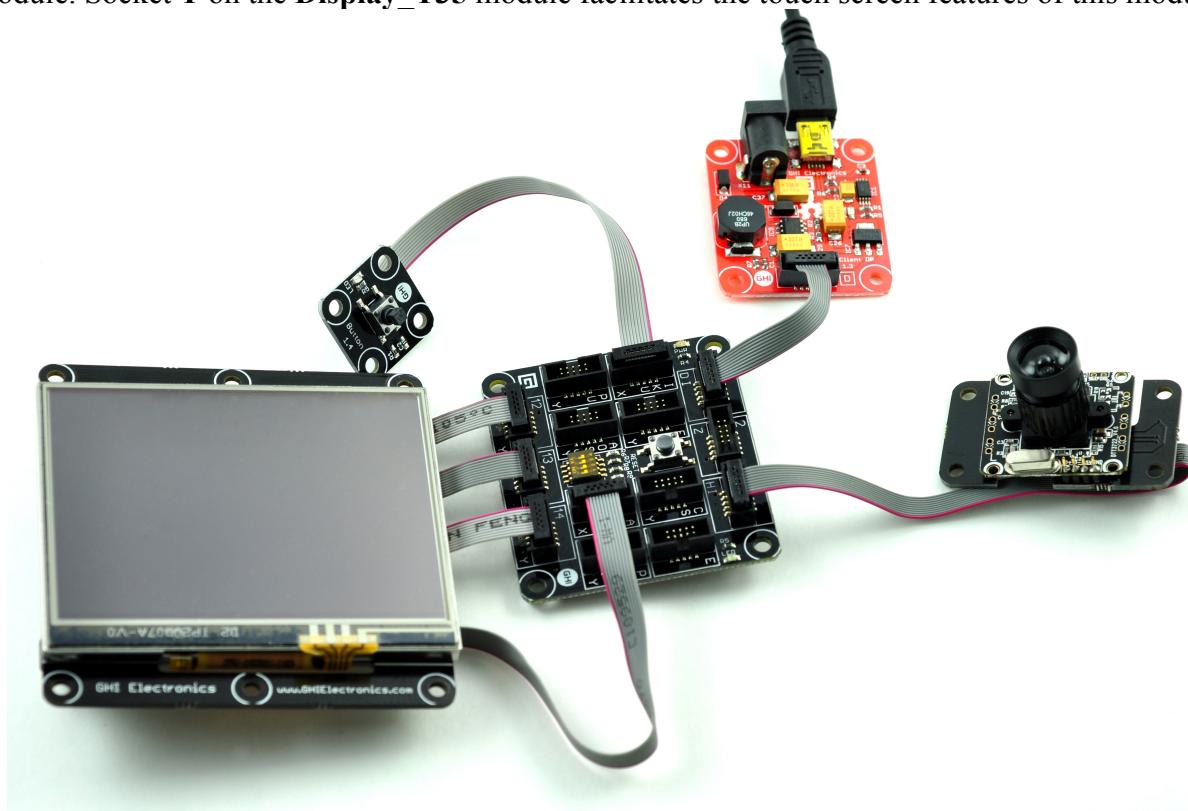


## Connecting the Modules to the Mainboard

Connect the **Button** and the **Camera** to the mainboard using module connector cables as described in the previous sections.

The .NET Gadgeteer designer can identify the sockets on modules and on the mainboard that are compatible. This method is described in the following section titled **Using the .NET Gadgeteer Designer UI**. In this example we will connect the remaining **Display\_T35** module manually.

The **Display\_T35** module has four connectors. Connect sockets 14, 13, and 12 to the **Display\_T35** module sockets labeled **R**, **G**, and **B**. As you might expect, these letters signify the color distribution of the **Display\_T35** module. Connect socket **10** on the mainboard to socket **T** on the **Display\_T35** module. Socket **T** on the **Display\_T35** module facilitates the touch screen features of this module.



### Warning

Make the actual connections between all modules and the mainboard before you connect the

USBClientDP to the USB port on your computer.

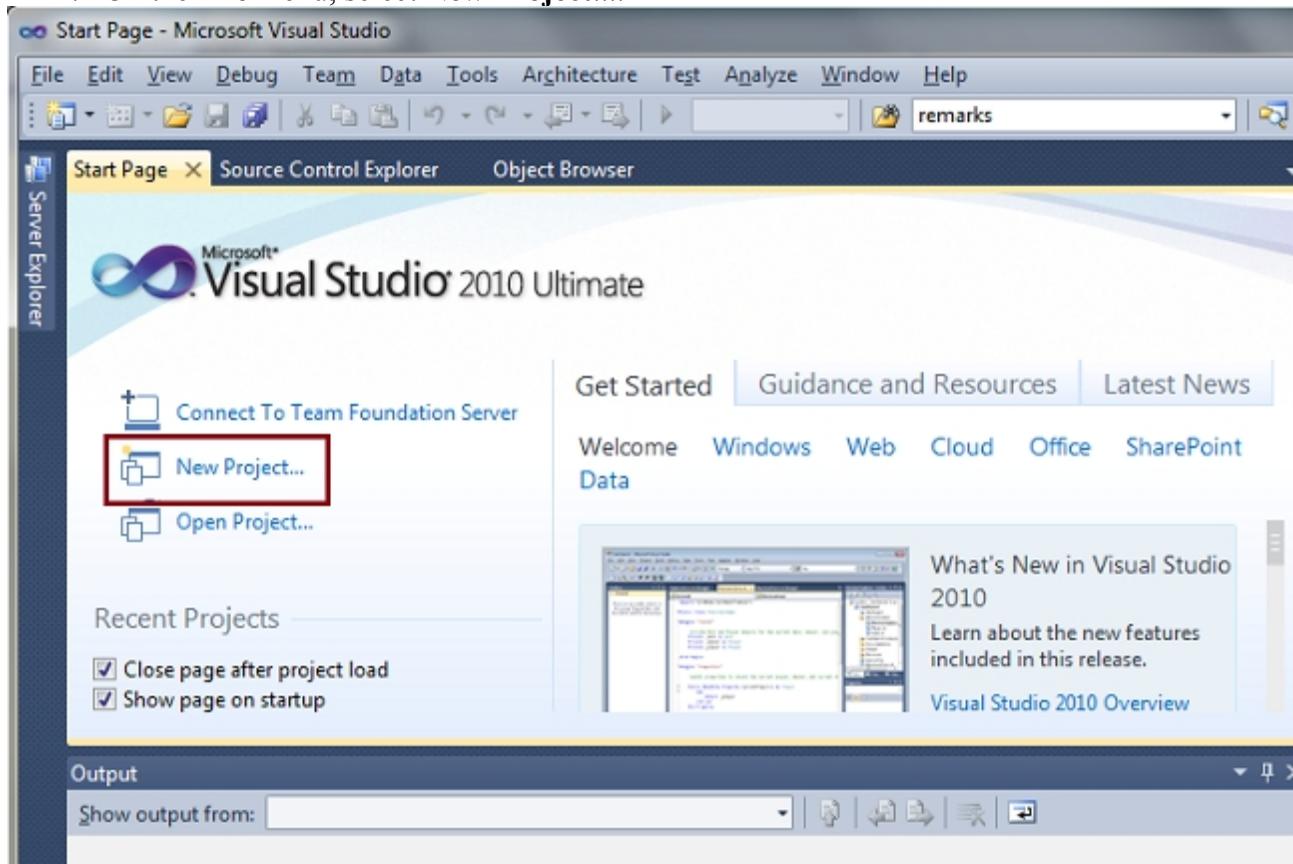
The following illustration shows the mainboard connected to a **Display\_T35** module, a **Button** module, a **Camera** module, and the **USBClientDP** module. Now, with all the other modules connected, you can connect the **USBClientDP** module to the USB port on your computer.

## ***Creating Your First .NET Gadgeteer Application.***

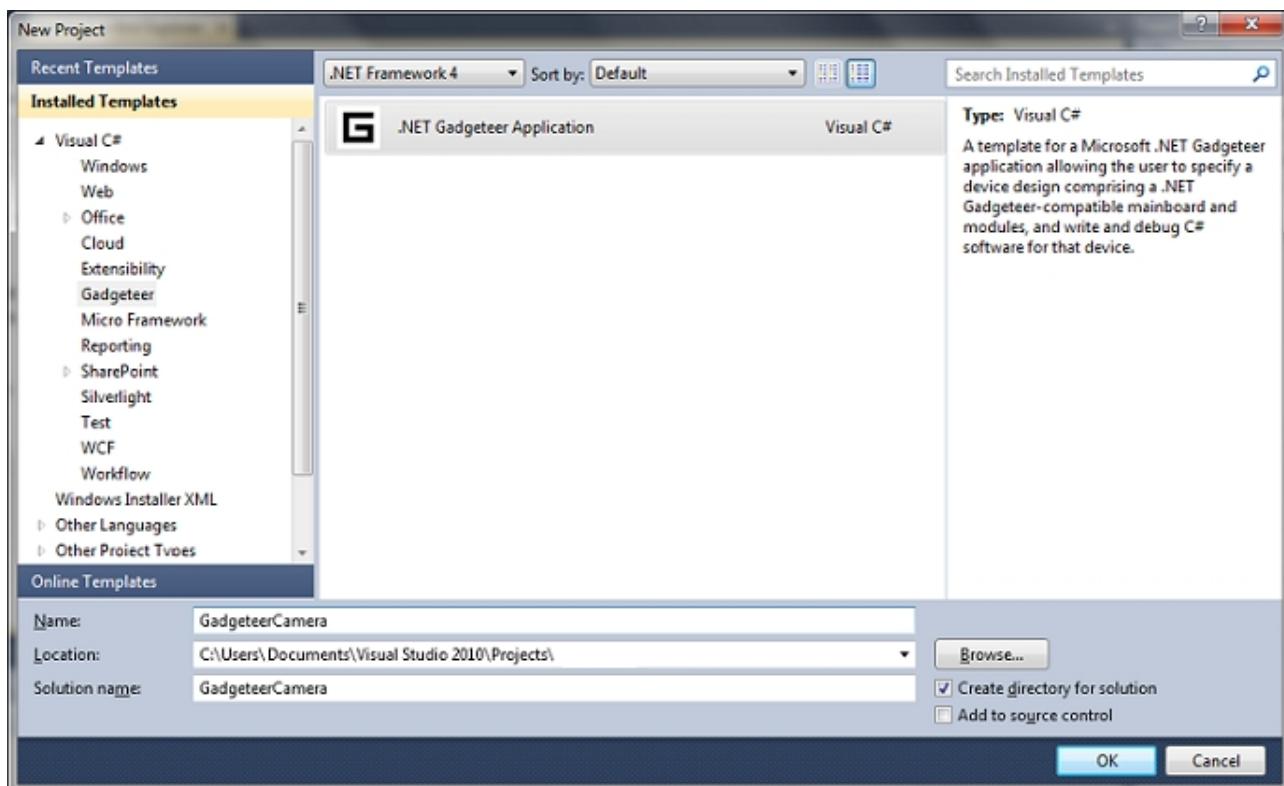
Start **Microsoft Visual Studio 2010** or **Microsoft Visual C# 2010 Express**. The following sequence will get your first .NET Gadgeteer Application up and running in less than half an hour.

### **To create a Visual Studio application:**

1. On the **File** menu, select **New Project....**

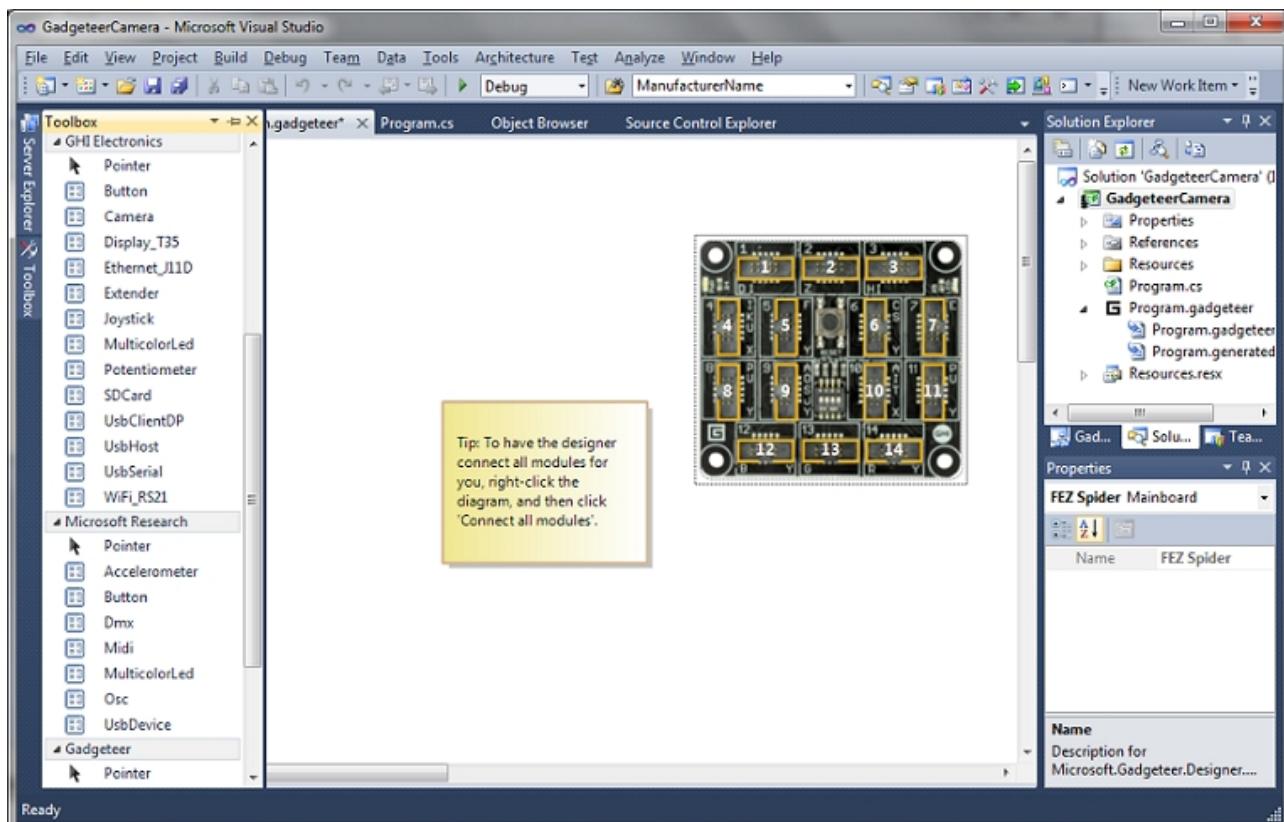


2. On the **New Project** screen, under **Installed Templates**, expand the **Visual C# category**.
3. Select **Gadgeteer**.
4. Select **.NET Gadgeteer Designer Application**. Name the project **GadgeteerCamera**.
5. Click **OK**.

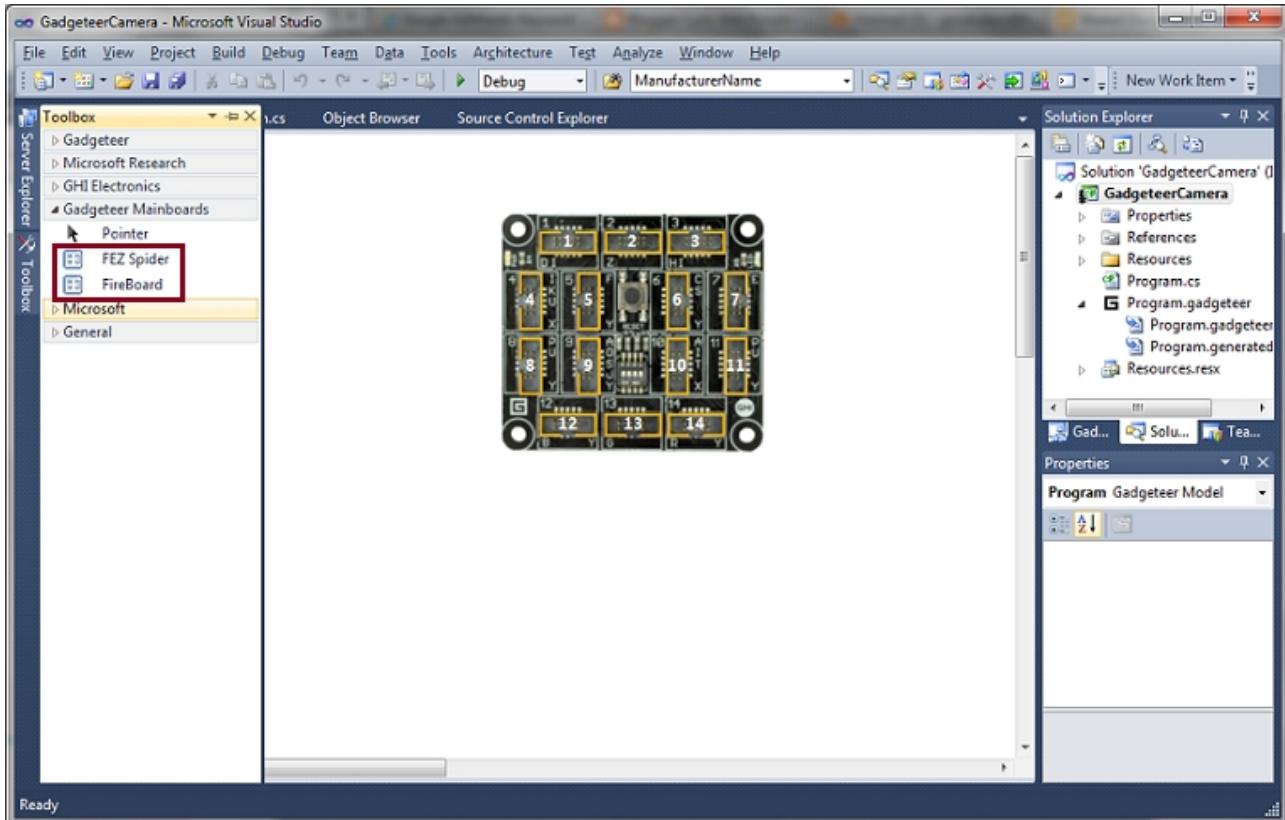


## Using the .NET Gadgeteer Designer UI

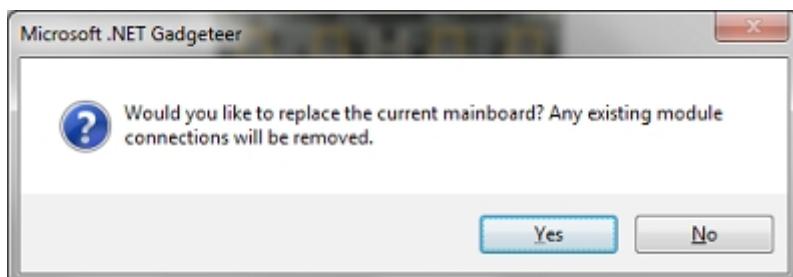
The .NET Gadgeteer Designer opens with the FEZ Spider Mainboard displayed on the canvas. If the **Toolbox** is not visible, click the **View** menu and select **Toolbox**. The **Toolbox** with a list of installed modules will open. You can resize or hide the **Toolbox** to make more work space on the canvas.



If you want to use a different mainboard, open the toolbox, and drag the mainboard icon of your choice onto the designer surface. The following illustration shows the open toolbox with two mainboard options.



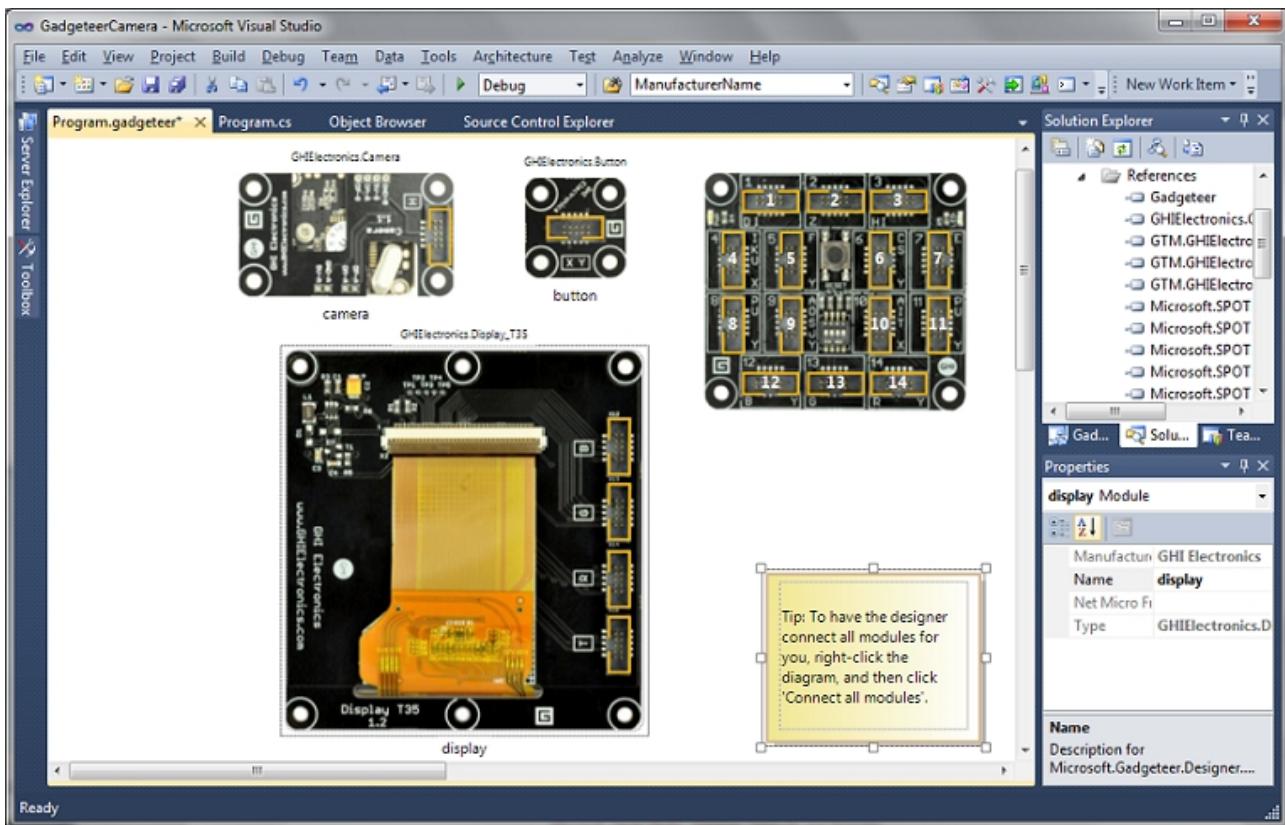
When you drag a new mainboard to the designer surface, you'll be prompted, as shown in the following illustration, to confirm replacement of the mainboard. All existing connections will be removed.



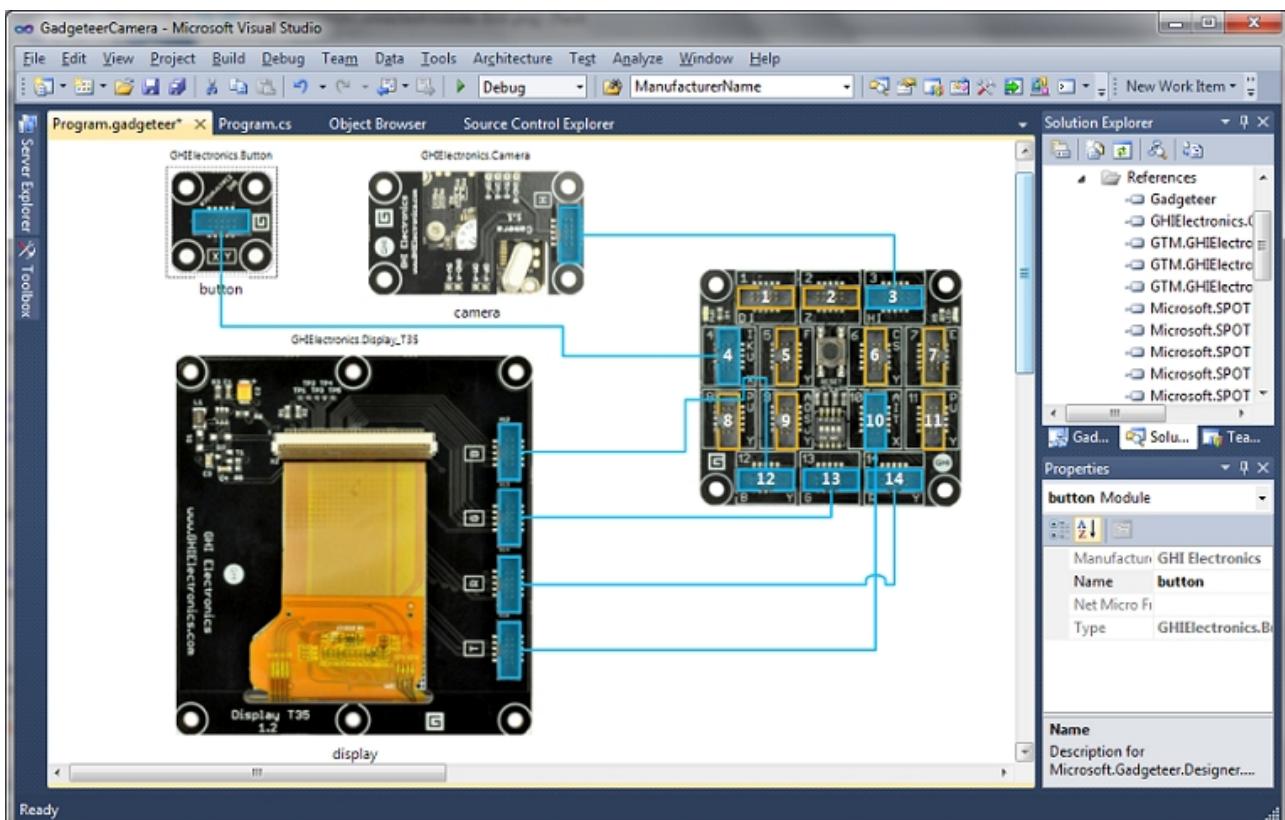
To continue building the example device, open the **Toolbox** and drag the modules for this example on to the work canvas. You will need the following modules:

- **Camera**
- **Display\_T35**
- **Button**

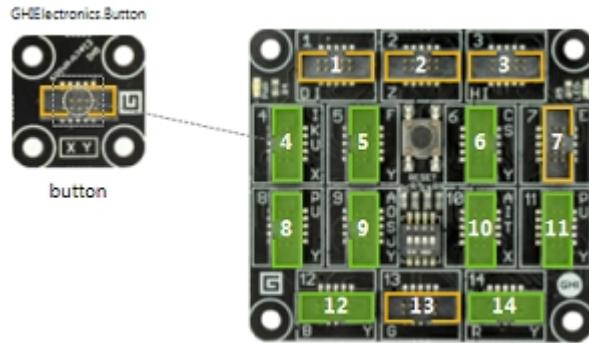
The Designer canvas with modules is shown in the following illustration.



As indicated by the instructions in the text box on the canvas, the .NET Gadgeteer Designer will graphically connect all the modules for you. Right click on the design surface and select **Connect all modules**. You can move the modules around on the design surface to make the connections easier to read. You can also delete any connection by right clicking on the connection and selecting **Delete**.



To manually set a connection, click on a socket in the diagram and hold down the left mouse button. Then you can drag a line that represents the connection from a module socket to a mainboard socket or in the opposite direction from the mainboard to a module. The sockets that the module can use will light up in green, as shown in the following illustration.



## Writing Code for Devices that use .NET Gadgeteer Modules

To specify what the modules should do in this application, now edit the **Program.cs** file. The following illustration shows the **Program.cs** file open in Visual Studio.

```

GadgeteerCamera - Microsoft Visual Studio
File Edit View Refactor Project Build Debug Team Data Tools Architecture Test Analyze Window Help
... Debug ManufacturerName ... New Work Item ...
Program.gadgeteer Program.cs X Object Browser Source Control Explorer
Program.cs
using Microsoft.SPOT;
using Microsoft.SPOT.Presentation;
using Microsoft.SPOT.Presentation.Controls;
using Microsoft.SPOT.Presentation.Media;

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

namespace GadgeteerCamera
{
    public partial class Program
    {
        void ProgramStarted()
        {
            // Access modules defined in the designer by typing their name:
            // e.g. button
            // camera1

            // Initialize event handlers here.
            // e.g. button.ButtonPressed += new GTM.MSR.Button.EventHandler(button_ButtonPressed);
            // ****

            // Do one-time tasks here
            Debug.Print("Program Started");
        }
    }
}

```

When using the designer, the modules are automatically instantiated by auto-generated code. This code can be found in the file **Program.Generated.cs**, but during normal use it is not necessary to view this file, and this file should not be edited because the Designer will automatically regenerate it and undo any direct changes made to it. The **Program.Generated.cs** file is shown in the following snippet.

```

// -----
// <auto-generated>
//   This code was generated by the Gadgeteer Designer.
//
//   Changes to this file may cause incorrect behavior and will be lost if
//   the code is regenerated.
// </auto-generated>
// -----

using Gadgeteer;
using GTM = Gadgeteer.Modules;

namespace GadgeteerCamera
{
    public partial class Program : Gadgeteer.Program
    {
        // GTM.Module defintions
        Gadgeteer.Modules.GHIElectronics.Button button;
        Gadgeteer.Modules.GHIElectronics.Camera camera;
        Gadgeteer.Modules.GHIElectronics.Display_T35 display;

        public static void Main()
        {
            //Important to initialize the Mainboard first
            Mainboard = new GHIElectronics.Gadgeteer.FEZSpider();

            Program program = new Program();
            program.InitializeModules();
            program.ProgramStarted();
            program.Run(); // Starts Dispatcher
        }

        private void InitializeModules()
        {
            // Initialize GTM.Modules and event handlers here.
            camera = new GTM.GHIElectronics.Camera(3);

            button = new GTM.GHIElectronics.Button(4);

            display = new GTM.GHIElectronics.Display_T35(14, 13, 12, 10);
        }
    }
}

```

## The Button Pressed Event and Delegate Method

Next, you generate code that will enable the program to react to a button press. To do this, assign an event handler for the [Button.ButtonPressed](#) event. The IntelliSense feature of Visual Studio makes this process very easy.

In the **Program.cs** file, after the comment that reads:

Initialize event handlers here.

Type `button.(button followed by a period)`. IntelliSense displays a list of properties, methods and events as shown in the following illustration.

The screenshot shows a code editor with C# code. A tooltip for the `button.ButtonPressed` event is open, displaying its documentation: "Raised when the state of Gadgeteer.Modules.GHIElectronics.Button is Pressed." Below the tooltip, the code shows the assignment of an event handler:

```
button.ButtonPressed += GTM.MSR.Button.ButtonEventHandler(button_ButtonPressed);
```

Using the arrow keys, select **ButtonPressed**. Then type **+=** (plus sign followed by an equals sign). IntelliSense offers option to automatically insert the rest of the declaration:

```
button.ButtonPressed +=  
    new Button.ButtonEventHandler(button_ButtonPressed); (Press TAB to insert)
```

Press **TAB** to confirm. IntelliSense offers to automatically generate a delegate method to handle the event:

```
button.ButtonPressed += new Button.ButtonEventHandler(button_ButtonPressed);  
Press TAB to generate handler 'button_ButtonPressed' in this class
```

Press **TAB** to confirm. The following code is generated:

```
void button_ButtonPressed(Button sender, Button.ButtonState state)  
{  
    throw new NotImplementedException();  
}
```

This event handler will be called each time the button is pressed. Delete the following line from the method:

```
throw new NotImplementedException();
```

And replace it with:

```
camera.TakePicture();
```

Now when the **Button** is pressed, the **Camera** module will take a picture.

```
void button_ButtonPressed(Button sender, Button.ButtonState state)  
{  
    camera.TakePicture();  
}
```

## The Picture Captured Event and Delegate Method

The **Camera** module raises an event when the **Camera** captures a picture. You can use the event to display the picture in the **Display\_T35** module.

The **PictureCaptured** event is an example of an asynchronous event. Calling **Camera.TakePicture** in the **Button.ButtonPressed** delegate starts the process, but the result of the action is not returned by the **ButtonPressed** delegate. Instead, the **GT.Picture** object returns as a parameter of the asynchronous **PictureCaptured** event.

When the **PictureCaptured** event occurs, you can get the **GT.Picture** object and display it by using the **SimpleGraphics** interface of the **Display\_T35** module. The **SimpleGraphics** interface supports the **DisplayImage** method, which accepts a **GT.Picture** object as a parameter, along with integer values to indicate the X and Y coordinates that position the image on the display.

All code required to display the picture is shown in the following implementation of the **PictureCaptured** event delegate.

```
void camera_PictureCaptured(Camera sender, GT.Picture picture)
{
    display.SimpleGraphics.DisplayImage(picture, 5, 5);
}
```

## Complete Application Code

All the code for a .NET Gadgeteer Application that takes a picture and displays it in the **Display\_T35** module is shown in the following example.

```

using Microsoft.SPOT;
using Microsoft.SPOT.Presentation;
using Microsoft.SPOT.Presentation.Controls;
using Microsoft.SPOT.Presentation.Media;

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

namespace GadgeteerCamera
{
    public partial class Program
    {
        void ProgramStarted()
        {
            // ****
            // Access modules defined in the designer by typing their name:
            //
            // e.g. button
            //       camera1

            Initialize event handlers here.
            e.g. button.ButtonPressed += new GTM.MSR.Button.EventHandler(button_ButtonPressed);
            ****

            // Do one-time tasks here

            button.ButtonPressed += new Button.ButtonEventHandler(button_ButtonPressed);
            camera.PictureCaptured += new Camera.PictureCapturedEventHandler(camera_PictureCaptured);

            Debug.Print("Program Started");
        }

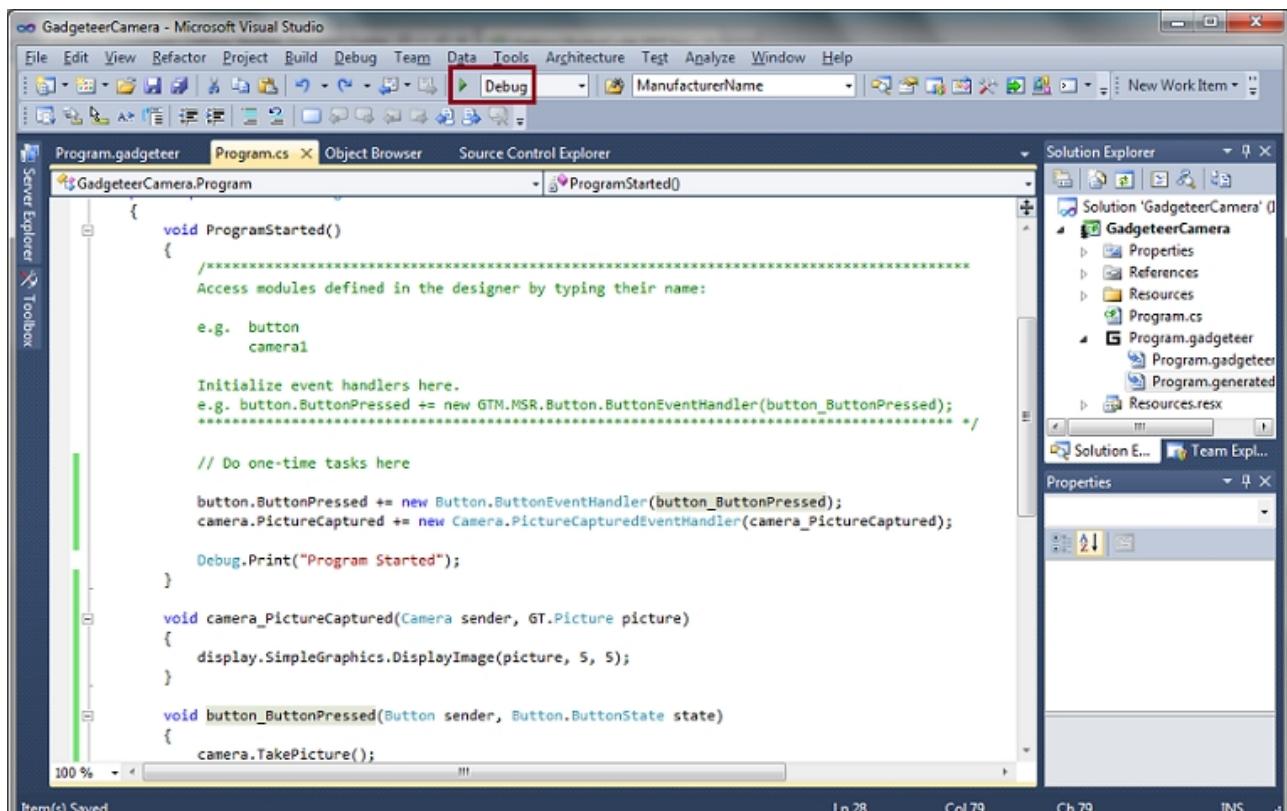
        void camera_PictureCaptured(Camera sender, GT.Picture picture)
        {
            display.SimpleGraphics.DisplayImage(picture, 5, 5);
        }

        void button_ButtonPressed(Button sender, Button.ButtonState state)
        {
            camera.TakePicture();
        }
    }
}

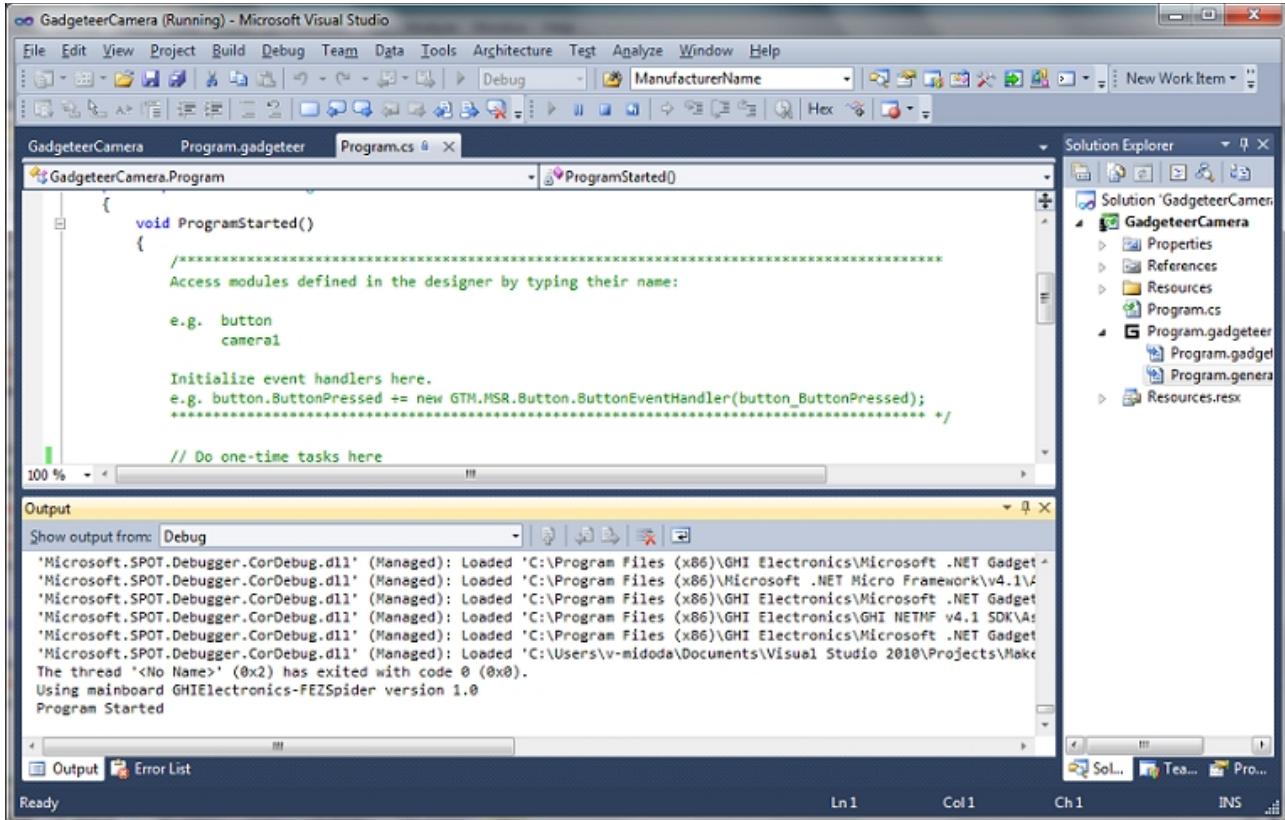
```

## Deploying Your .NET Gadgeteer Application

To deploy this application to a mainboard and begin running it, select **Start Debugging** from the **Debug** menu, or press **F5**.



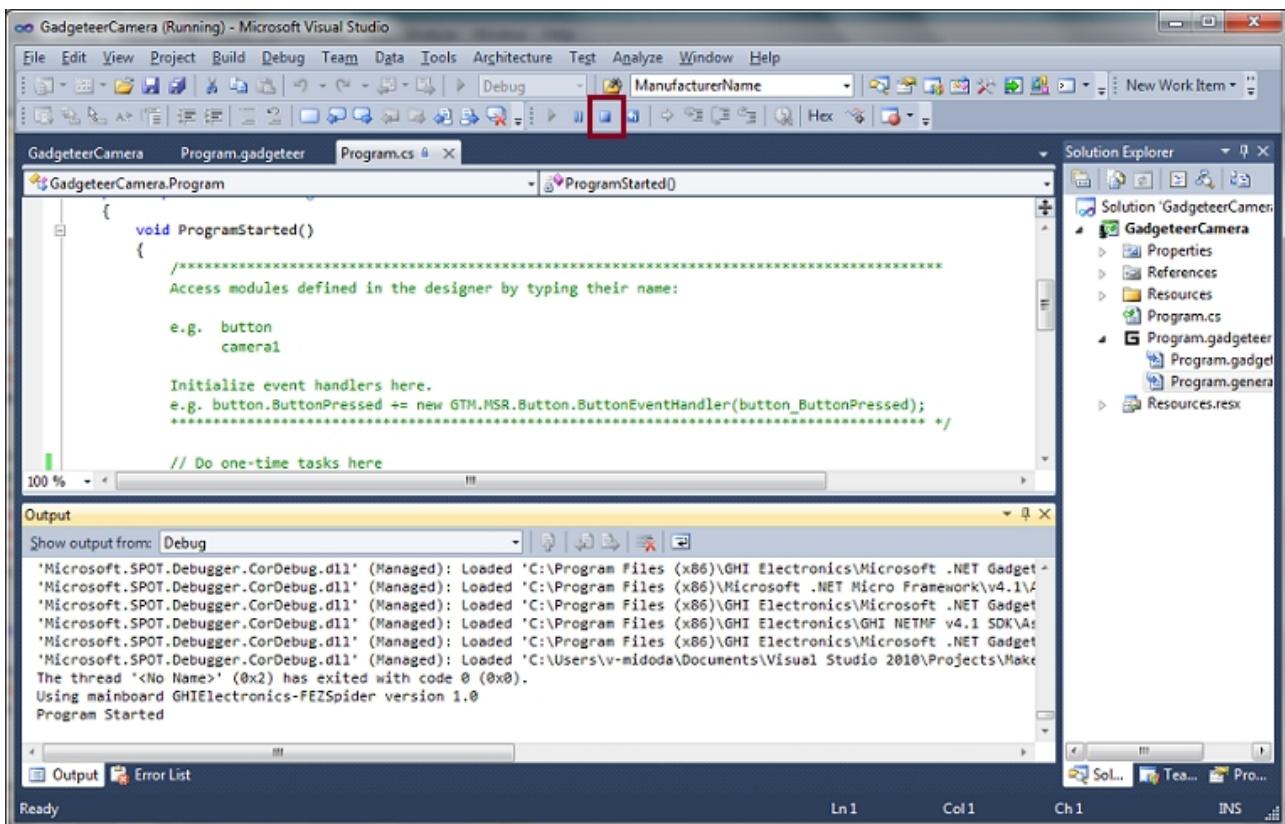
Make sure that the **Output Window** is visible by pressing the **CTRL + ALT + O** key combination on your keyboard. If you have enabled sounds, you should hear Windows make the "USB disconnected" sound, followed by the "USB connected" sound as the Mainboard reboots. The **Output Window** should show the process of loading various files and assemblies. The final line, which appears once the application begins to run, should read **Program Started**.



## Running the .NET Gadgeteer Device

When you see **Program Started** appear on the Output window, you can take a picture. Smile for the **Camera**, and push the **Button**. Your picture will appear on the screen of the **Display\_T35** module. If it works - congratulations! You have completed your first .NET Gadgeteer application.

To exit debugging mode, select **Stop Debugging** from the **Debug** menu, or press **Shift + F5**.



## Adding Timed Actions for a Surveillance Camera

An interesting extension of the camera application in the previous example is programming the application to take pictures automatically at an interval set by an instance of the [GT.Timer](#) class.

To create an instance of the [GT.Timer](#) class, add the following global variable to the [Program](#) class, as shown in the following example. This line of code initializes the [GT.Timer](#) to raise the [Tick](#) event at an interval of 2000 milliseconds (2 seconds).

```
GT.Timer timer = new GT.Timer(2000);
```

Create the delegate to handle the [GT.Timer.Tick](#) event, but stop the timer until the button is pushed. The following code shows the set-up in the [ProgramStarted](#) method.

```
timer.Tick += new GT.Timer.TickEventHandler(timer_Tick);
timer.Stop();
```

In the [ButtonPressed](#) handler replace [Camera.TakePicture](#) with the following code. The new code starts and stops the [GT.Timer.Tick](#) event and toggles the LED indicator on the [Button](#). When the [GT.Timer](#) is firing events, you can use the event handler to take pictures instead of the [ButtonPressed](#) handler.

```
void button_ButtonPressed(Button sender, Button.ButtonState state)
{
    if (button.IsLedOn)
        timer.Stop();
    else
        timer.Start();

    button.ToggleLED();
}
```

The implementation of the [GT.Timer.Tick](#) event is shown in the following example.

```
void timer_Tick(GT.Timer timer)
{
    camera.TakePicture();
}
```

When the LED is off and the user pushes the **Button**, the LED indicator turns on and remains on while the Camera module takes pictures at two second intervals. When the user pushes the button again, the [GT.Timer](#) stops, and the LED turns off.

The following code contains all the code for the camera with timer. The boxes show the changes to the previous example.

```

using Microsoft.SPOT;
using Microsoft.SPOT.Presentation;
using Microsoft.SPOT.Presentation.Controls;
using Microsoft.SPOT.Presentation.Media;

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

namespace GadgeteerCamera
{
    public partial class Program
    {
        GT.Timer timer = new GT.Timer(2000);

        void ProgramStarted()
        {
            //***** Access modules defined in the designer by typing their name:
            //e.g. button
            //camera1

            //Initialize event handlers here.
            //e.g. button.ButtonPressed += new GT.MSR.Button.ButtonEventHandler(button_ButtonPressed); ****
            //****

            // Do one-time tasks here

            button.ButtonPressed += new Button.ButtonEventHandler(button_ButtonPressed);
            camera.PictureCaptured += new Camera.PictureCapturedEventHandler(camera_PictureCaptured);

            timer.Tick += new GT.Timer.TickEventHandler(timer_Tick);
            timer.Stop();

            Debug.Print("Program Started");
        }

        void timer_Tick(GT.Timer timer)
        {
            camera.TakePicture();
        }

        void camera_PictureCaptured(Camera sender, GT.Picture picture)
        {
            display.SimpleGraphics.DisplayImage(picture, 5, 5);
        }

        void button_ButtonPressed(Button sender, Button.ButtonState state)
        {
            if (button.IsLedOn)
                timer.Stop();
            else
                timer.Start();

            button.ToggleLED();
        }
    }
}

```

## Troubleshooting

This section describes common issues that you may encounter and provides suggestions for how to fix them.

### Set-up Issues

If VS is running when you install GadgeteerCore, you'll need to close and restart it before creating your first project. Otherwise you won't see the .NET Gadgeteer Application template.

You may need to change the USB name of the target mainboard in your first project. The most efficient way to do this is using the [MFDDeploy tool](#).

Sometimes VS will hang at the display: "The debugging target is not in an initialized state;

rebooting". Push the reset button on the mainboard to fix this.

If you're using the Display\_T35 and see a null reference exception on startup, verify that the touch socket is connected both in the designer and on the module.

If you're using a laptop and you see errors on deployment like "Please check your hardware", try plugging a 7 volt DC power supply into the USBClientDP module.

## Compile Time Errors

If you receive compilation errors when you attempt to deploy and run your application, read the error message carefully. Most errors fall into one of two categories:

- Syntax: A statement is missing required syntax, for example, the ending semi-colon character. These types of errors are generally reported unambiguously in the Error output window. Fix the syntax problem and try again.
- Identifier: An identifier is unknown or invalid. This can happen if you spell the identifier incorrectly, or do not qualify it correctly. All identifiers are case sensitive; for example, Button cannot be entered as button. To help avoid problems with identifiers, use the IntelliSense feature of Visual Studio. IntelliSense presents only valid identifiers.

## Unexpected Application Behavior

If your application does not behave as expected (for example, pressing the button does not raise the event), start by checking that the physical socket to which the hardware module is connected agrees with the initialization in code of the identifier that corresponds to the module.

For example, if you connect a **Button** to mainboard socket 4, but initialize it to socket 8, the button will not work.

```
// Button is actually plugged into socket 4.  
button = new GTM.GHIElectronics.Button(8);
```

The programming model for the .NET Gadgeteer platform is event driven. Events are raised that correspond to a hardware change or physical action. For example, when you press a button, the ButtonPressed event is raised, and when you release it, the ButtonReleased event is raised.

You can use debug statements inside your event handlers to make sure that your handler is receiving the event. For example, if your LED does not light when you press the button, you can insert a statement inside the event handler for the ButtonPressed event to make sure that your button is in fact receiving the event.

```
private void Button_ButtonPressed(GTM.Button sender, GTM.Button.ButtonState state)  
{  
    Debug.Print("Button Pressed");  
    camera.TakePicture();  
}
```

When you deploy and run your application, check the Visual Studio Debug Output window for your message. If the message does not appear at the expected time (for example, when you press the button), make sure that the physical socket and logical initializer are in agreement, as previously

described. If they are, the button or the module connector cable might be defective. Unplug the mini USB cable from your computer, swap the module connector cable or the button with another from your hardware kit, reconnect the mini USB cable, and try again.

## Deployment

Occasionally, you may receive an error as you attempt to deploy your application to a mainboard. This can happen if the mainboard is not connected to your computer, or the mainboard requires a restart. If the mainboard is disconnected, connect it and retry. If the mainboard is connected when this happens, disconnect it from your computer (by unplugging the mini USB cable), wait a few seconds, and reconnect it. Then try the deployment again.

## Device Drivers

When you install the .NET Gadgeteer core, the device drivers that are needed to communicate with a mainboard are also installed. This process usually does not require any intervention on your part.

In some cases, the .NET Gadgeteer core installation or kit installation might not install the device drivers automatically. If your computer is having problems communicating with a mainboard that you suspect are related to the device drivers, please refer to the [Tiny CLR Forum](#) or to the kit's support forum.