Getting Started with the FEZ Hydra Kit for Microsoft .NET Gadgeteer

Welcome to the FEZ Hydra Kit from GHI Electronics. This kit is compatible with Microsoft .NET Gadgeteer. The FEZ Hydra 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 Hydra 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 Hydra 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:

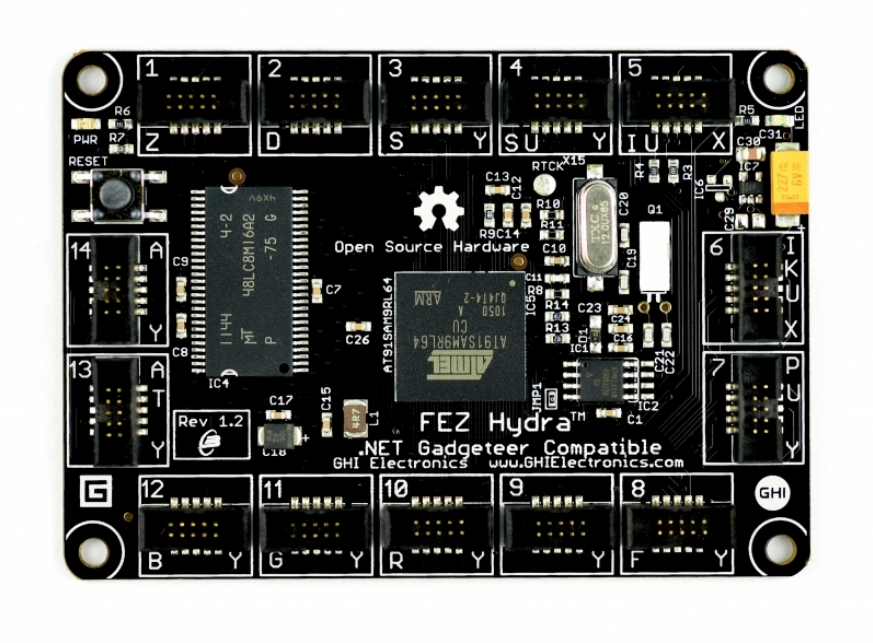
- FEZ Hydra Mainboard
- A Red **USB Client Single Power (USBClientSP)** module
- A **Joystick** module
- A **LED7R** module
- A **LightSense** module
- Module connector cables

The FEZ Hydra Mainboard

The FEZ Hydra Mainboard is the most important part of the GHI Electronics FEZ Hydra Kit. The FEZ Hydra 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 Hydra mainboard by these sockets allow you to extend the FEZ Hydra mainboard with communication, user interaction, sensing, and actuation capabilities.

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



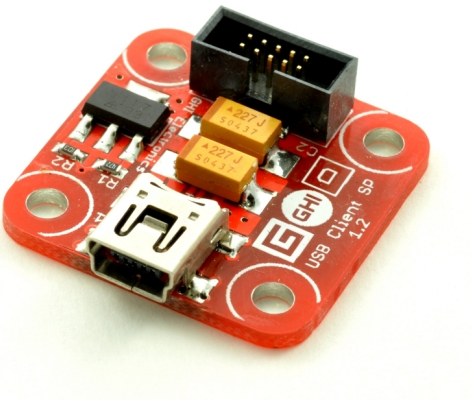
The USB Client Single Power Device Module

The **USB Client Single Power (USBClientSP)** module (colored red) enables you to connect the FEZ Hydra Mainboard to your computer for programming and debugging. The single-powered module is itself powered by a USB port on a computer. A **USBClientSP** module supplies power to the FEZ Hydra and to any other modules that are connected to it.

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

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

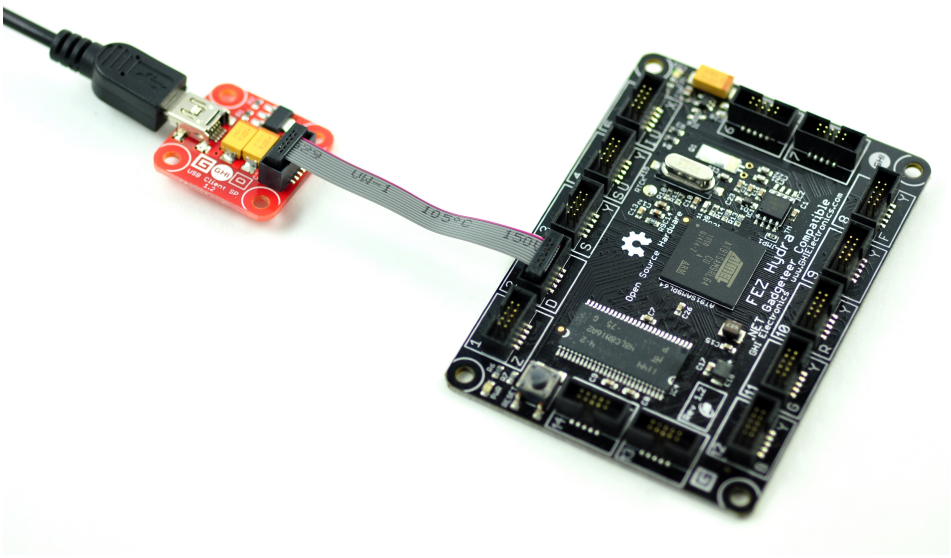
In a similar way, all .NET Gadgeteer-compatible modules have letters next to their sockets that identify which mainboard sockets they can be connected to. Many modules are labeled with multiple letters. This means that they can be connected to **any** of the labeled sockets.



The Module Connector Cable

Your hardware kit includes enough module connector cables to get you started. These cables are all identical and can be used interchangeably to connect modules to the FEZ Hydra 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 **2** on the **FEZHydra 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 **USBClientSP** module. However, **do not connect the other end to your computer yet.**



Warning

When plugging or unplugging any module into a FEZ Hydra 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 Hydra; if you plug or unplug a module on the FEZ Hydra while it is powered, the hardware could be damaged.

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

Turn the **Joystick** module over. Next to the connector is the letter **A**. This means that a **Joystick** module can be connected to one of the sockets labeled **A** on a mainboard.

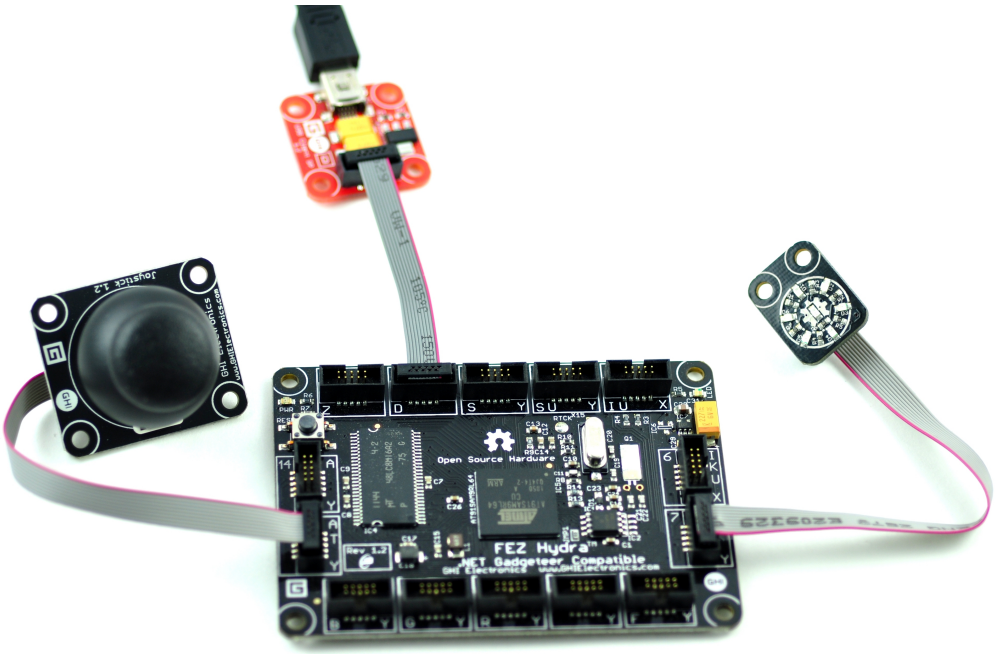
Get a **LED7R** module from your hardware kit. Turn the **LED7R** module over. This module has a single connector labeled **Y**. Multiple Sockets on the **FEZHydra** mainboard support modules labeled with the letter **Y**. Plug one end of a connector cable to the socket on the **LED7R** module and the other end to the **FEZHydra** on any socket labeled **Y**.

Connecting the Modules to the Mainboard

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 two previously mentioned modules, **Joystick** and **LED7R** manually.

Warning
Make the actual connections between all modules and the mainboard before you connect the USBClientSP to the USB port on your computer.

The following illustration shows the mainboard connected to a **Joystick** module, a **LED7R** module, and the **USBClientSP** module. Now, with all the other modules connected, you can connnect the **USBClientSP** module to the USB port on your computer. The **Joystick** will be connected to socket **13**, and the **LED7R** will be connected to socket **7.**

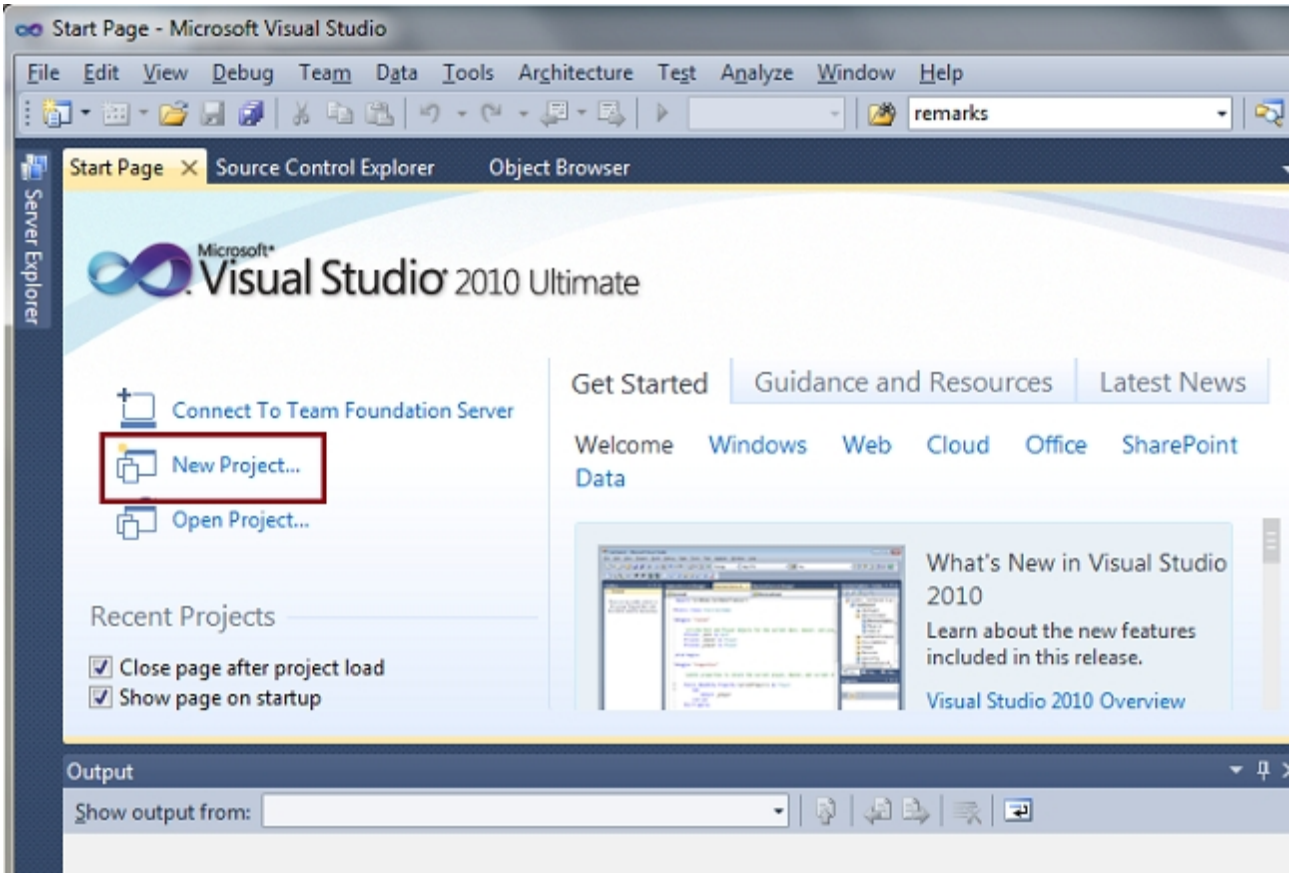


Creating Your First .NET Gadgeteer Application.

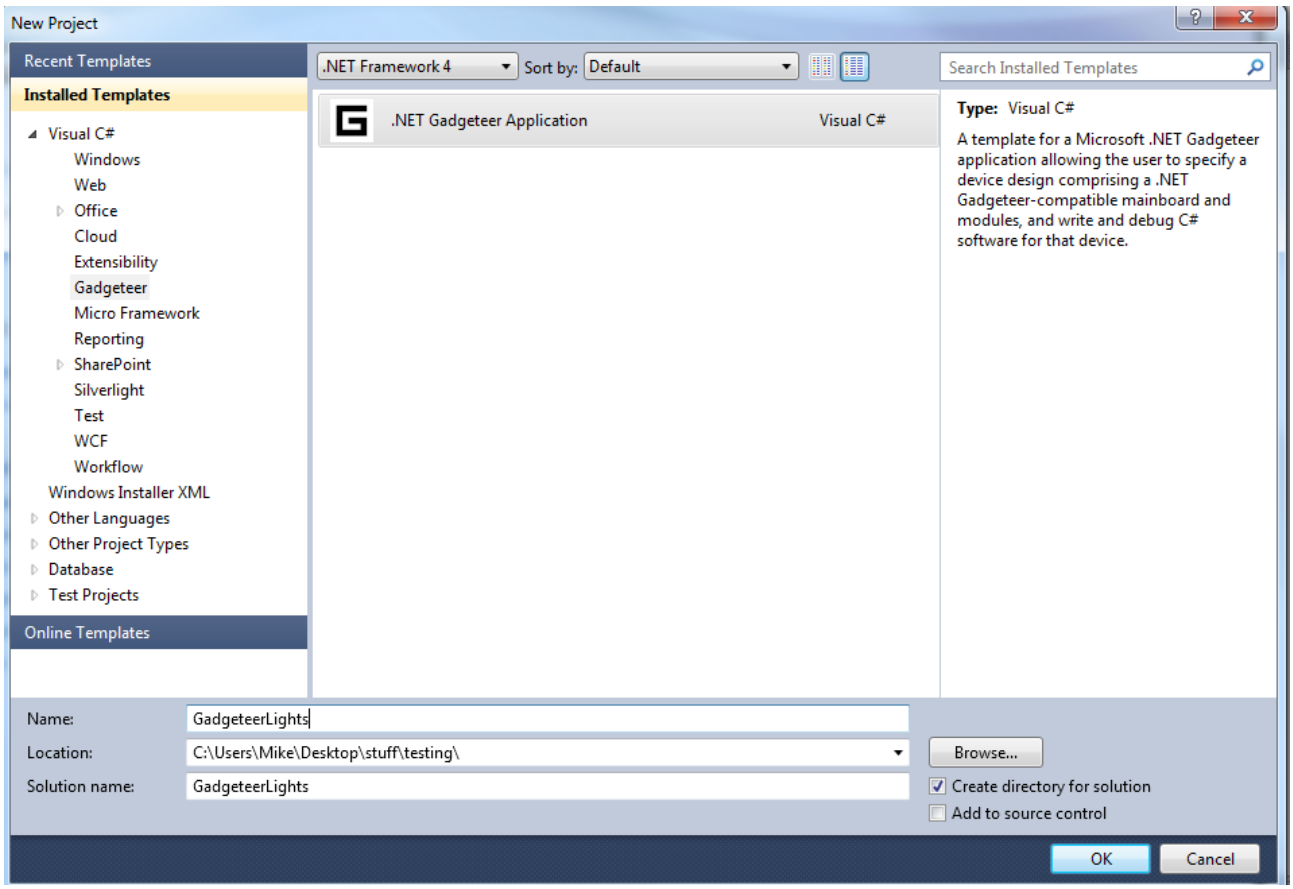
Start **Microsoft Visual Studio 2010** or **Microsoft Visual C# 2010 Express**. The following sequence will get your first .NET Gadgeteer Application up and running in less than half an hour.

To create a Visual Studio application:

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



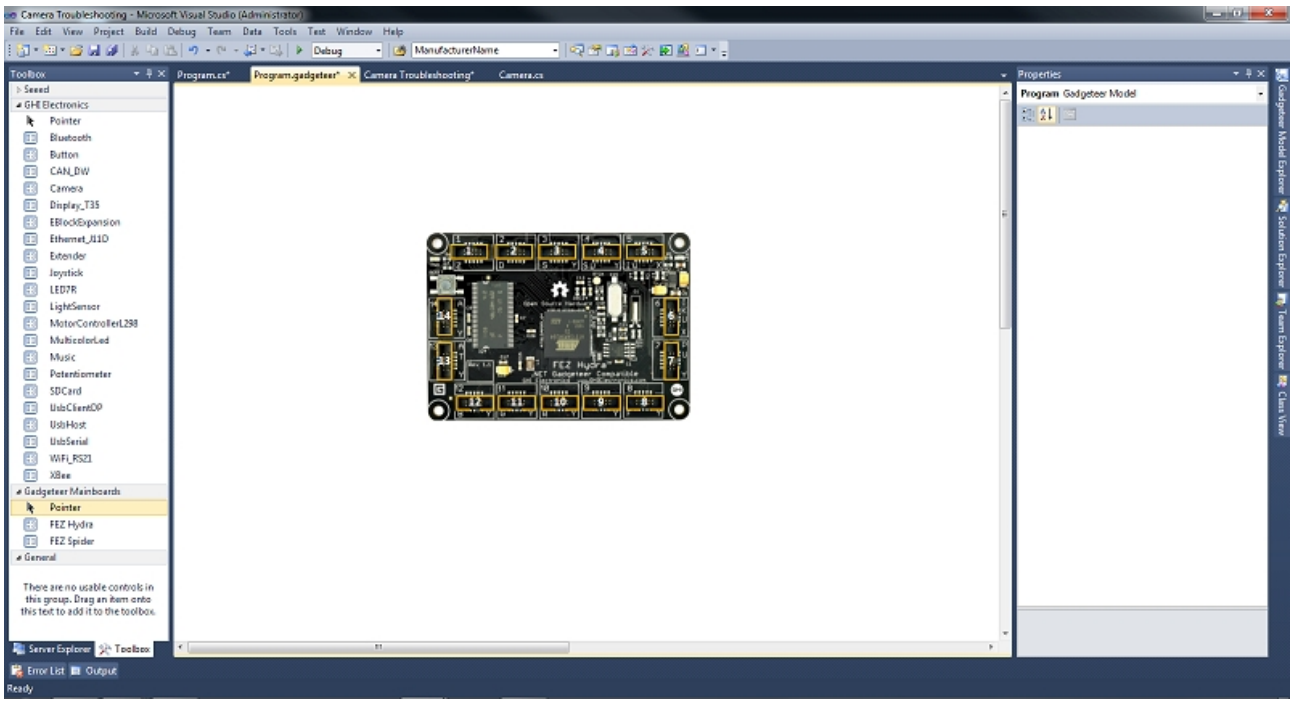
2. On the **New Project** screen, under **Installed Templates**, expand the **Visual C#** category.
3. Select **Gadgeteer**.
4. Select **.NET Gadgeteer Designer Application**. Name the project **GadgeteerLights**.



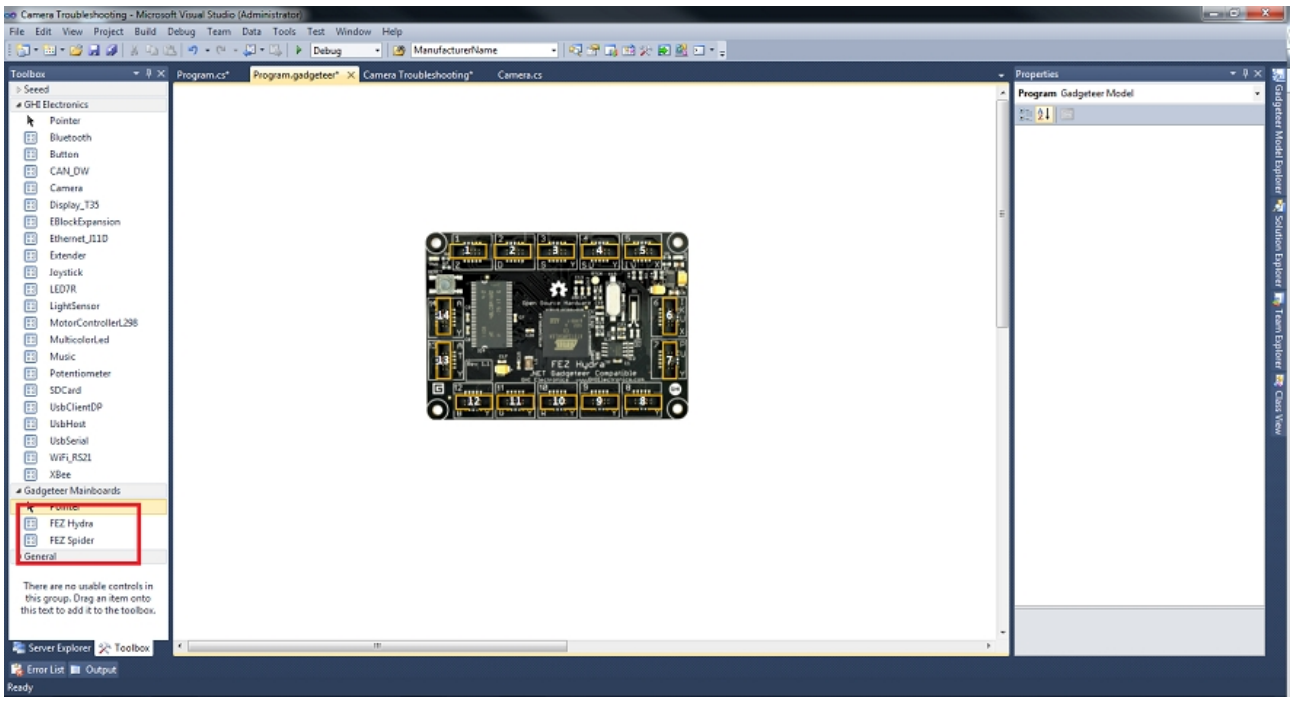
5. Click **OK**.

Using the .NET Gadgeteer Designer UI

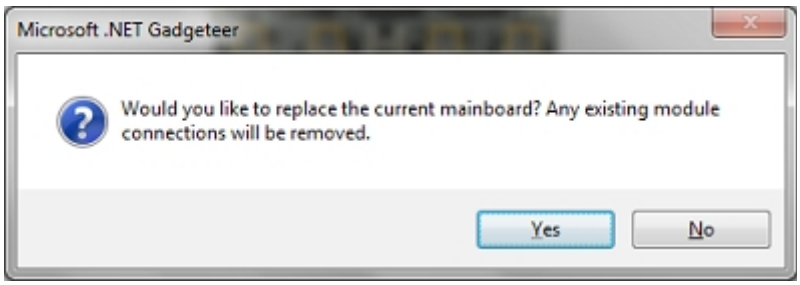
The **.NET Gadgeteer Designer** opens with the **FEZHydra Mainboard** displayed on the canvas. If the **Toolbox** is not visible, click the **View** menu and select **Toolbox**. The **Toolbox** with a list of installed modules will open. You can resize or hide the **Toolbox** to make more work space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



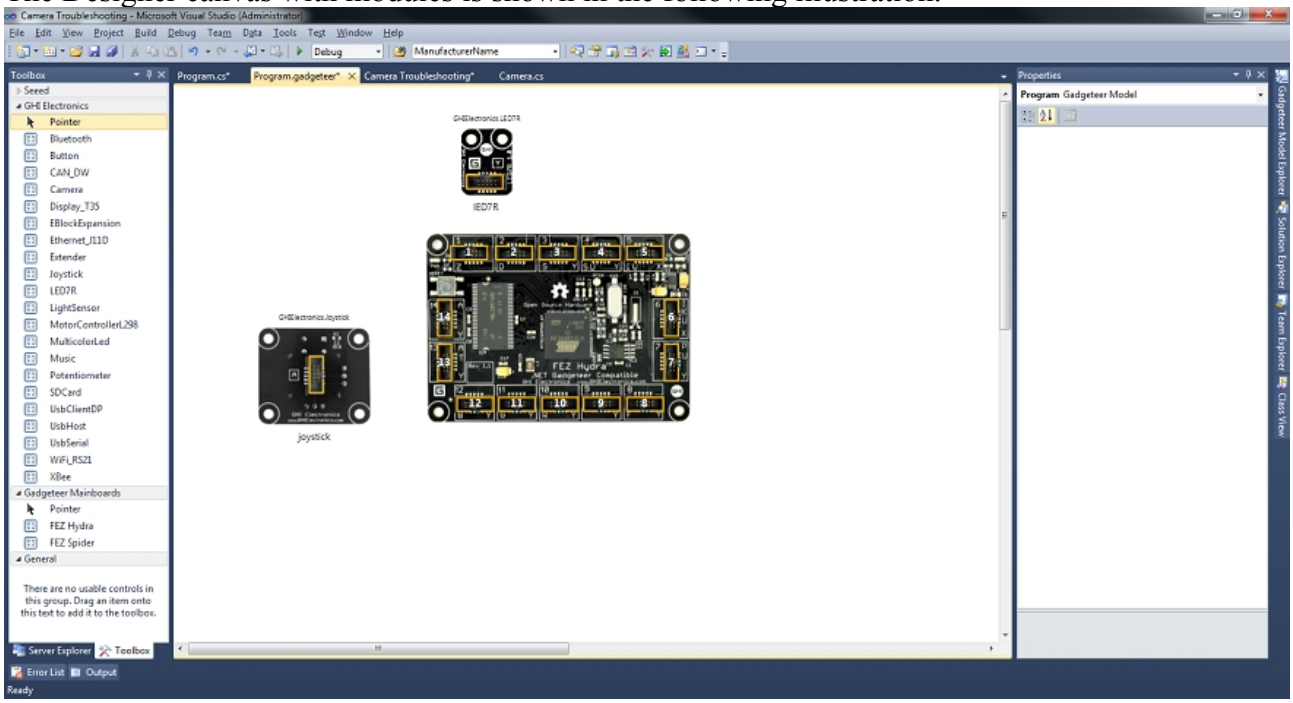
When you drag a new mainboard to the designer surface, you'll be prompted, as shown in the following illustration, to confirm replacement of the mainboard. All existing connections will be removed.



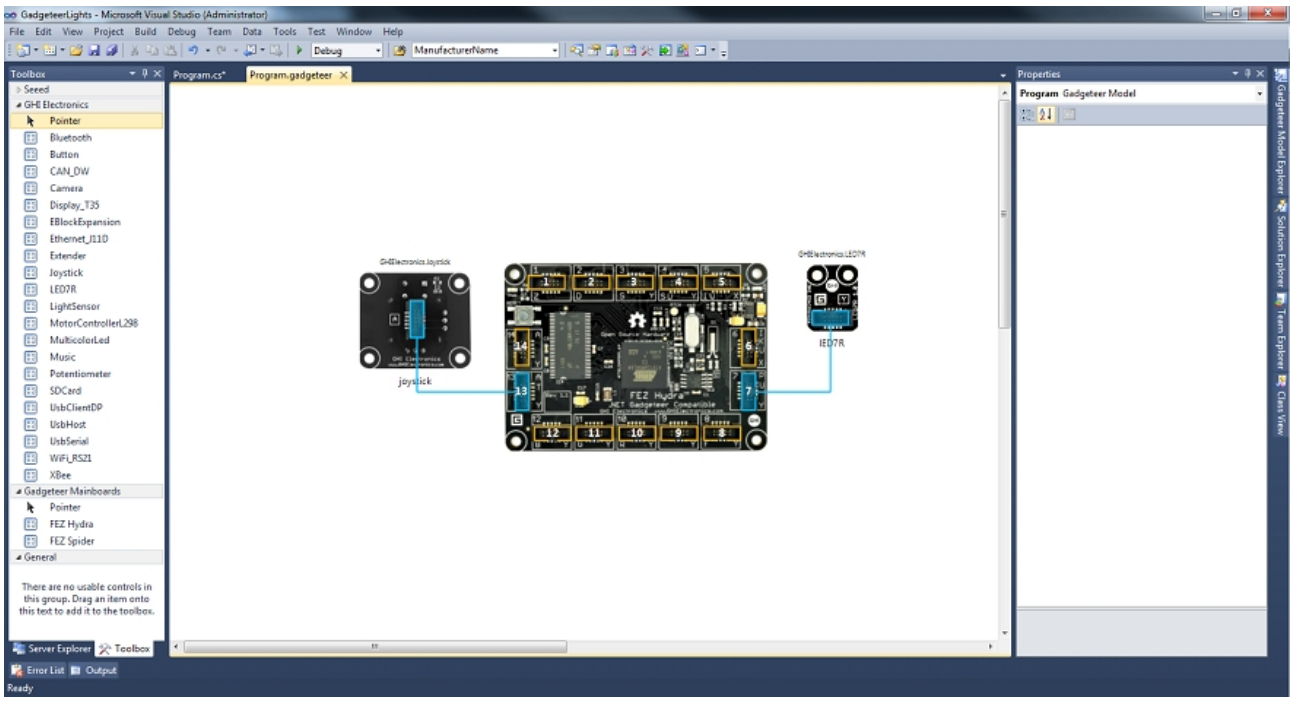
To continue building the example device, open the **Toolbox** and drag the modules for this example on to the work canvas. You will need the following modules:

- Joystick
- LED7R

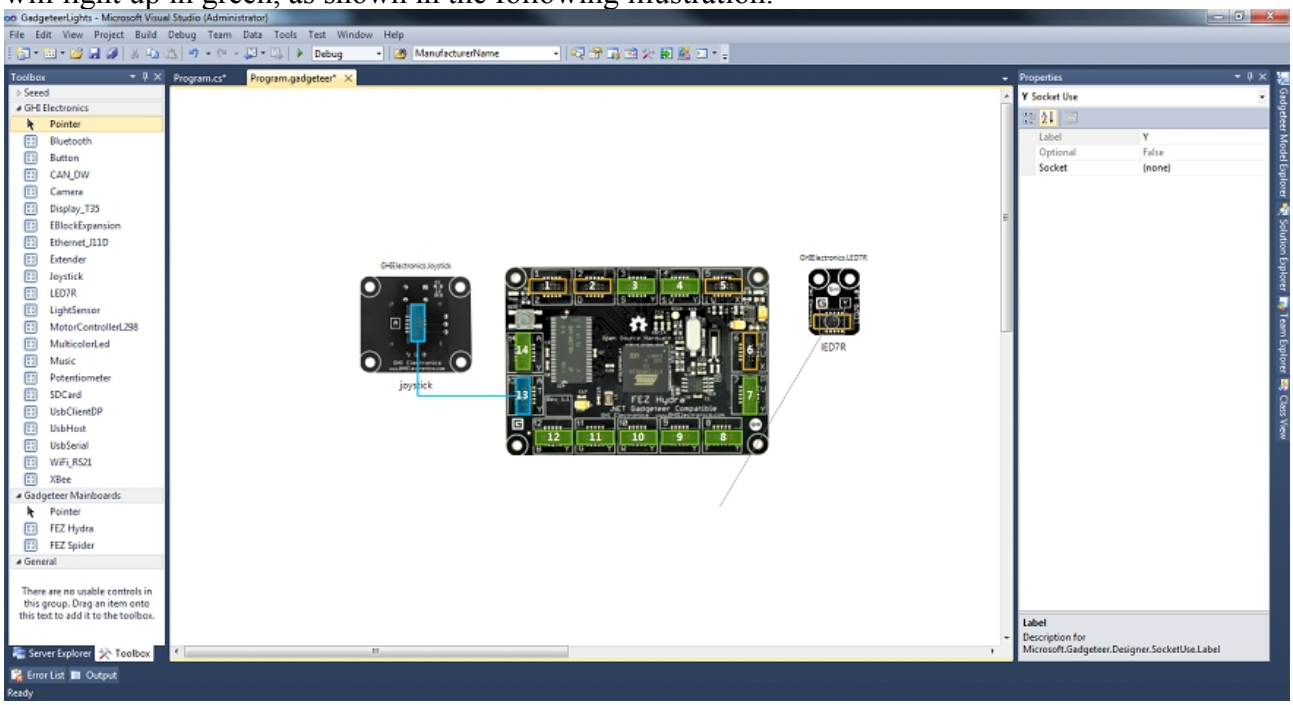
The Designer canvas with modules is shown in the following illustration.



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**. Make sure both modules are connected as in the diagram below.

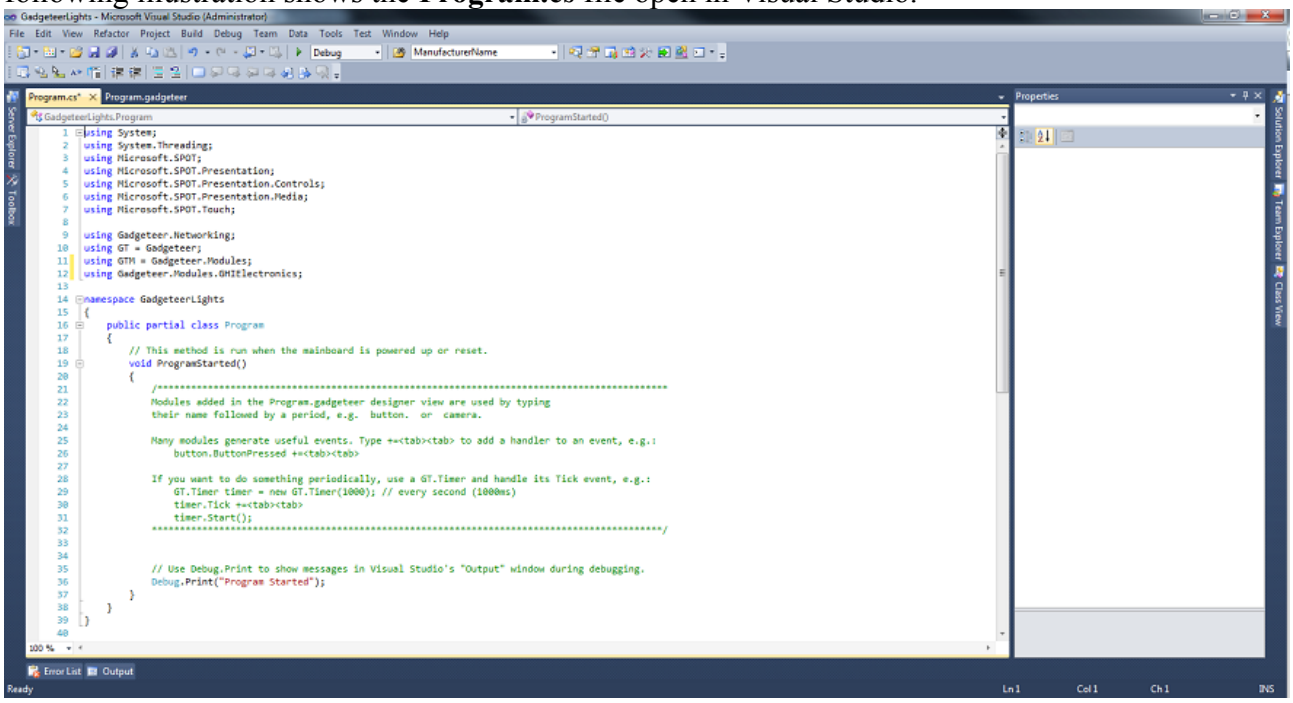


To manually set a connection, click on a socket in the diagram and hold down the left mouse button. Then you can drag a line that represents the connection from a module socket to a mainboard socket or in the opposite direction from the mainboard to a module. The sockets that the module can use will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

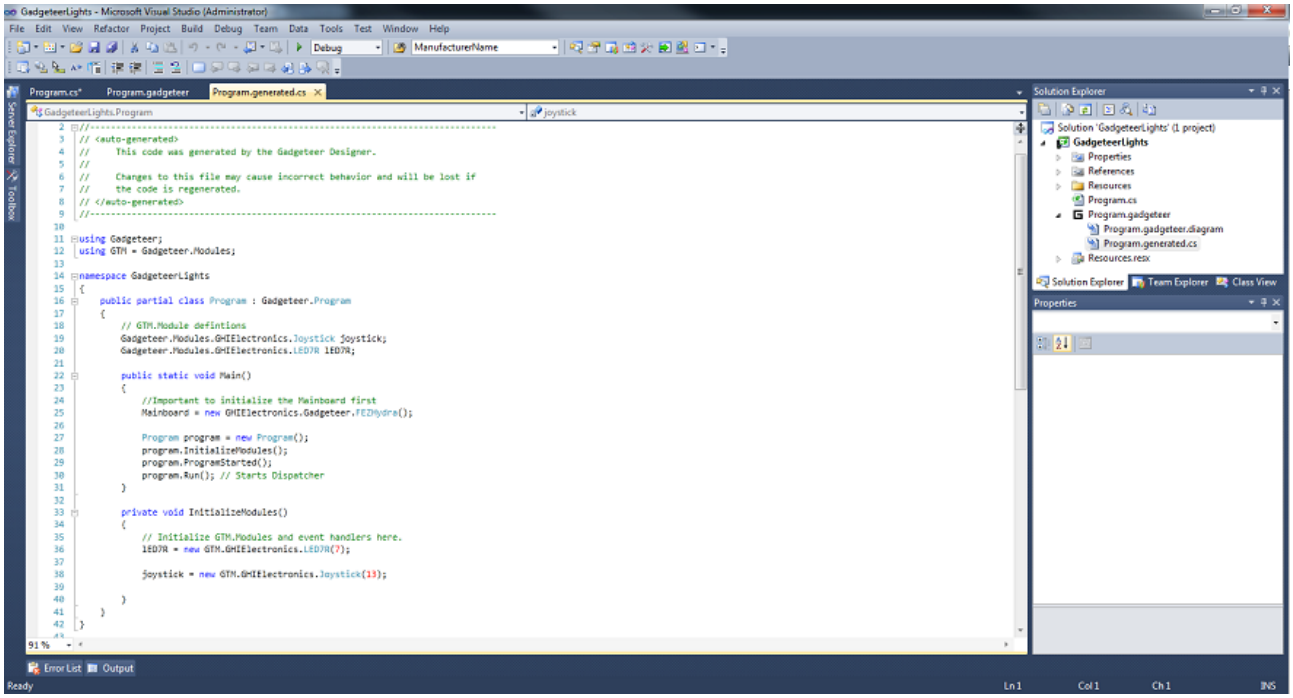


Writing Code for Devices that use .NET Gadgeteer Modules

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.



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.



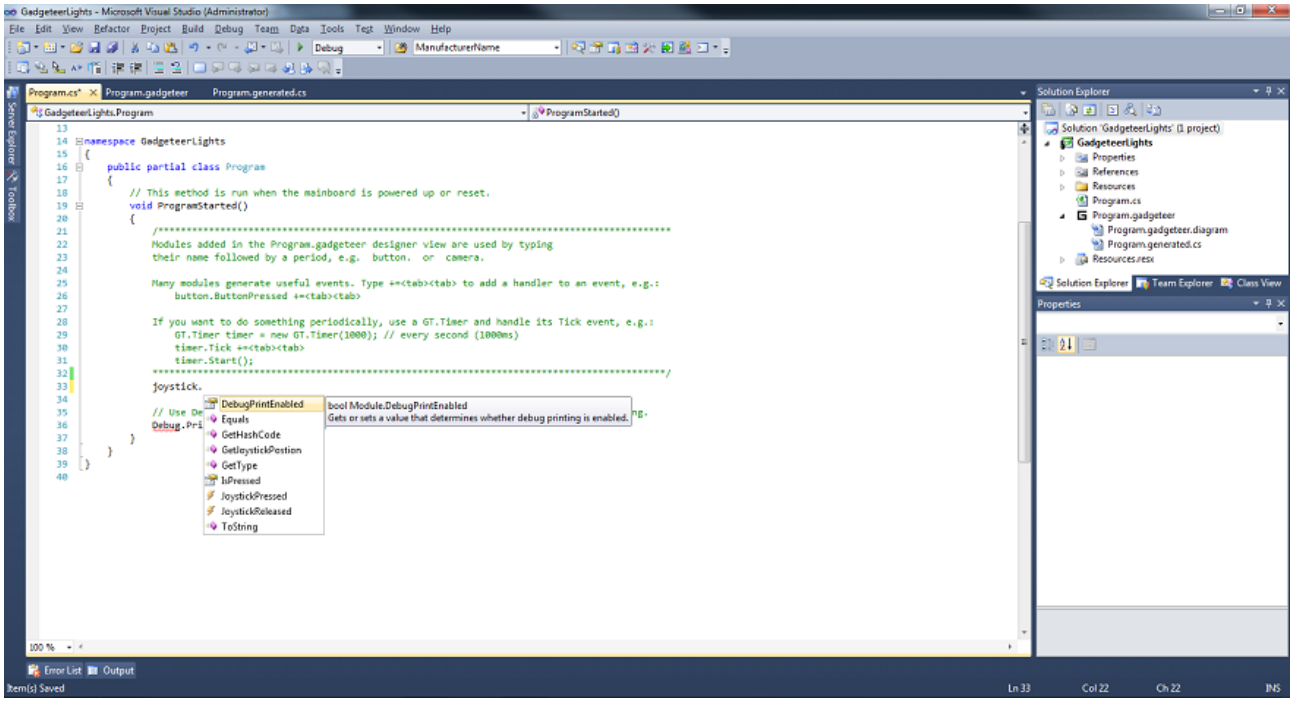
The Joystick Button Pressed Event and Delegate Method

Next, you generate code that will enable the program to react to a button press. To do this, assign an event handler for the Joystick.JoystickPressed **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 joystick.(joystick followed by a period). IntelliSense displays a list of properties, methods and events as shown in the following illustration.



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

```
joystick.JoystickPressed +=

// Use Debug.Print to show
Debug.Print("Program Started");
```

new Joystick.JoystickEventHandler(joystick_JoystickPressed); (Press TAB to insert)

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

```
joystick.JoystickPressed +=new Joystick.JoystickEventHandler(joystick_JoystickPressed);

// Use Debug.Print to show
Debug.Print("Program Started");
```

Press TAB to generate handler 'joystick_JoystickPressed' in this class

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

```
void joystick_JoystickPressed(Joystick sender, Joystick.JoystickState state)
{
    throw new NotImplementedException();
}
```

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

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

And replace it with:

```
Debug.Print("Joystick Pressed");
lED7R.TurnLightOff(7);
```

This will be explained in a moment. Repeat the same process for Joystick Released, including this code in the handler.

```
Debug.Print("Joystick Released");
lED7R.TurnLightOn(7, true);
```

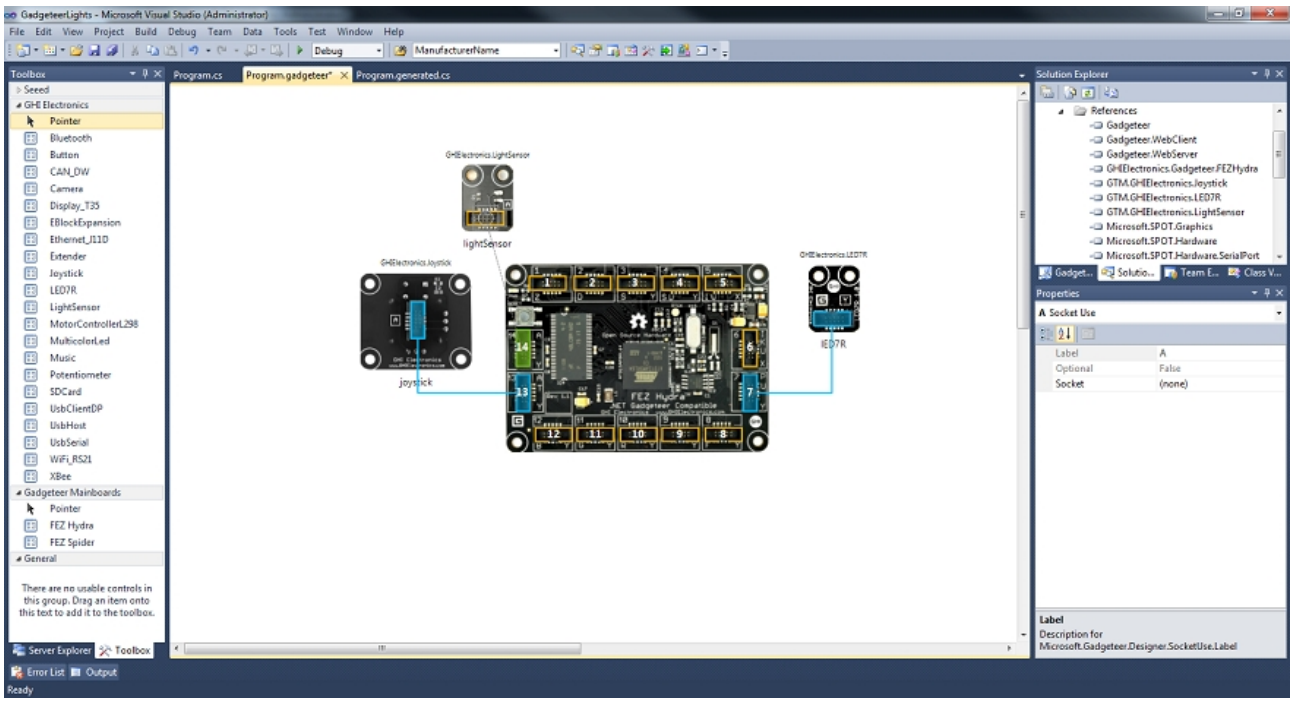
Now when the **Joystick** button is pressed and released, the **LED7R** module will flash the middle red light.

```
33              joystick.JoystickPressed += new Joystick.JoystickEventHandler(joystick_JoystickPressed);
34              joystick.JoystickReleased += new Joystick.JoystickEventHandler(joystick_JoystickReleased);
35
36              // Use Debug.Print to show messages in Visual Studio's "Output" window during debugging.
37              Debug.Print("Program Started");
38          }
39
40          void joystick_JoystickReleased(Joystick sender, Joystick.JoystickState state)
41          {
42              Debug.Print("Joystick Released");
43              lED7R.TurnLightOn(7, true);
44          }
45
46          void joystick_JoystickPressed(Joystick sender, Joystick.JoystickState state)
47          {
48              Debug.Print("Joystick Pressed");
49              lED7R.TurnLightOff(7);
50          }
```

The LightSense Module

The **LightSense** module measures the current exposure to light. You can use the JoystickPressed event to display the current exposure to light.

We will connect this module in the same way we did with the previous two. This module requires socket type **A**, so the only remaining socket available is socket 14. Connet the module both physically and in the designer, then save the file.



When the JoystickPressed event occurs, you can get the light exposure percentage and display it by using the following code:

```
Debug.Print("Light Sensor %: " + lightSensor.ReadLightSensorPercentage());
```

```
void joystick_JoystickPressed(Joystick sender, Joystick.JoystickState state)
{
    Debug.Print("Joystick Pressed");

    Debug.Print("Light Sensor %: " + lightSensor.ReadLightSensorPercentage());
    lED7R.TurnLightOff(7);
}
```

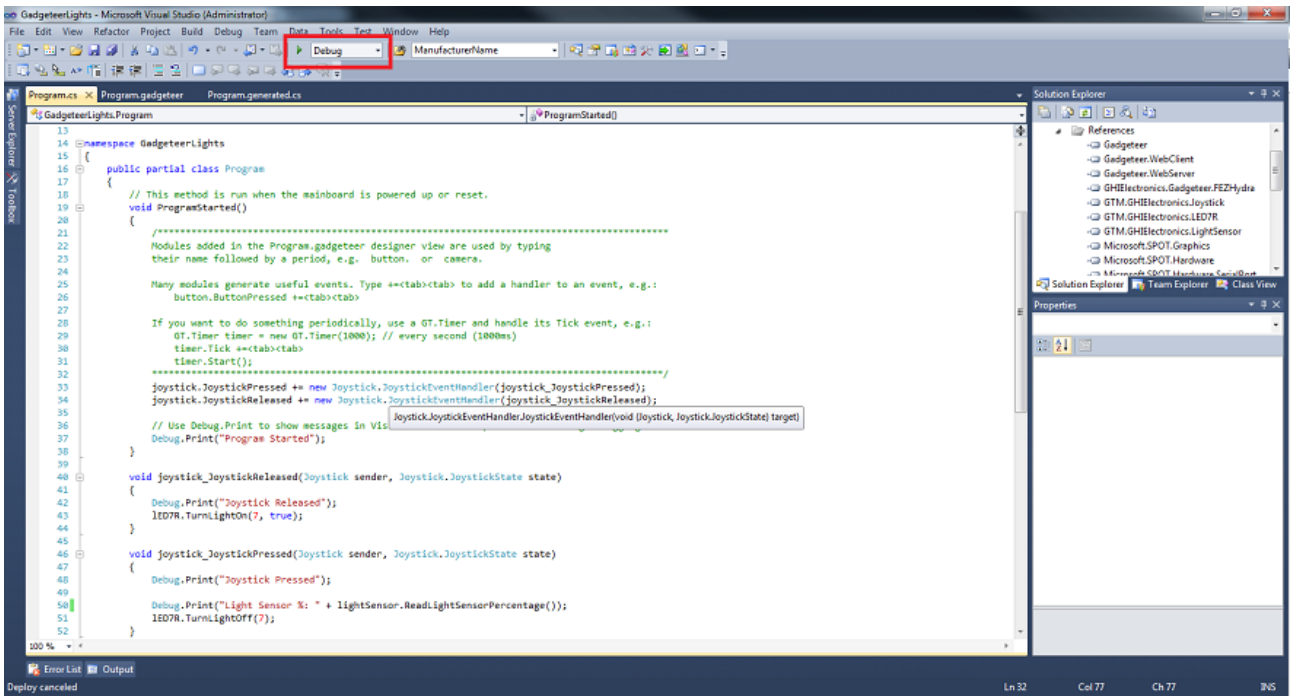
Complete Application Code

All the code for a .NET Gadgeteer Application that reads the Joystick button state, turns a light on and off, and reads the LightSense light exposure percentage is shown in the following example.

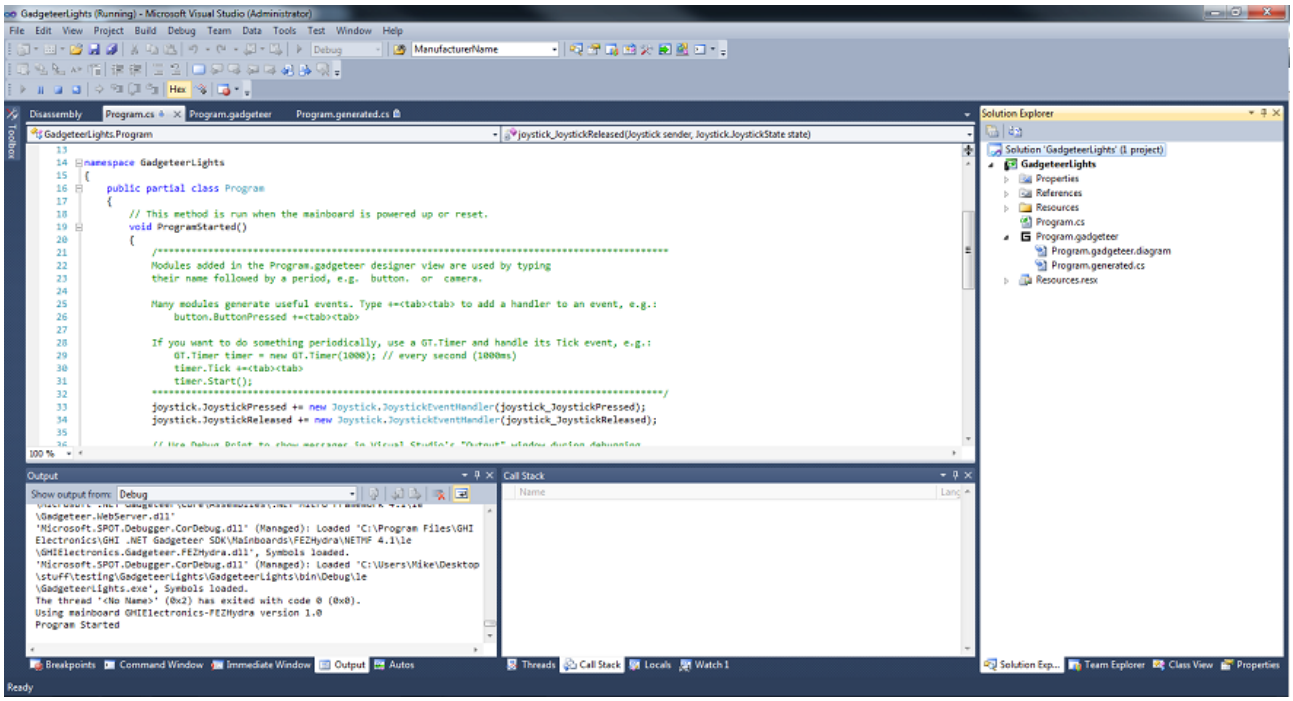
```
using System;
using System.Threading;
using Microsoft.SPOT;
using Microsoft.SPOT.Presentation;
using Microsoft.SPOT.Presentation.Controls;
using Microsoft.SPOT.Presentation.Media;
using Microsoft.SPOT.Touch;

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

namespace GadgeteerLights
{
    public partial class Program
    {
        // This method is run when the mainboard is powered up or reset.
        void ProgramStarted()
        {
            /*******************************************************************************************
            Modules added in the Program.gadgeteer designer view are used by typing
            their name followed by a period, e.g.  button.  or  camera.

            Many modules generate useful events. Type +=<tab><tab> to add a handler to an event, e.g.:
                button.ButtonPressed +=<tab><tab>

            If you want to do something periodically, use a GT.Timer and handle its Tick event, e.g.:
                GT.Timer timer = new GT.Timer(1000); // every second (1000ms)
                timer.Tick +=<tab><tab>
                timer.Start();
            *******************************************************************************************/
            joystick.JoystickPressed += new Joystick.JoystickEventHandler(joystick_JoystickPressed);
            joystick.JoystickReleased += new Joystick.JoystickEventHandler(joystick_JoystickReleased);

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

        void joystick_JoystickReleased(Joystick sender, Joystick.JoystickState state)
        {
            Debug.Print("Joystick Released");
            lED7R.TurnLightOn(7, true);
        }

        void joystick_JoystickPressed(Joystick sender, Joystick.JoystickState state)
        {
            Debug.Print("Joystick Pressed");

            Debug.Print("Light Sensor %: " + lightSensor.ReadLightSensorPercentage());
            lED7R.TurnLightOff(7);
        }
    }
}
```

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**.



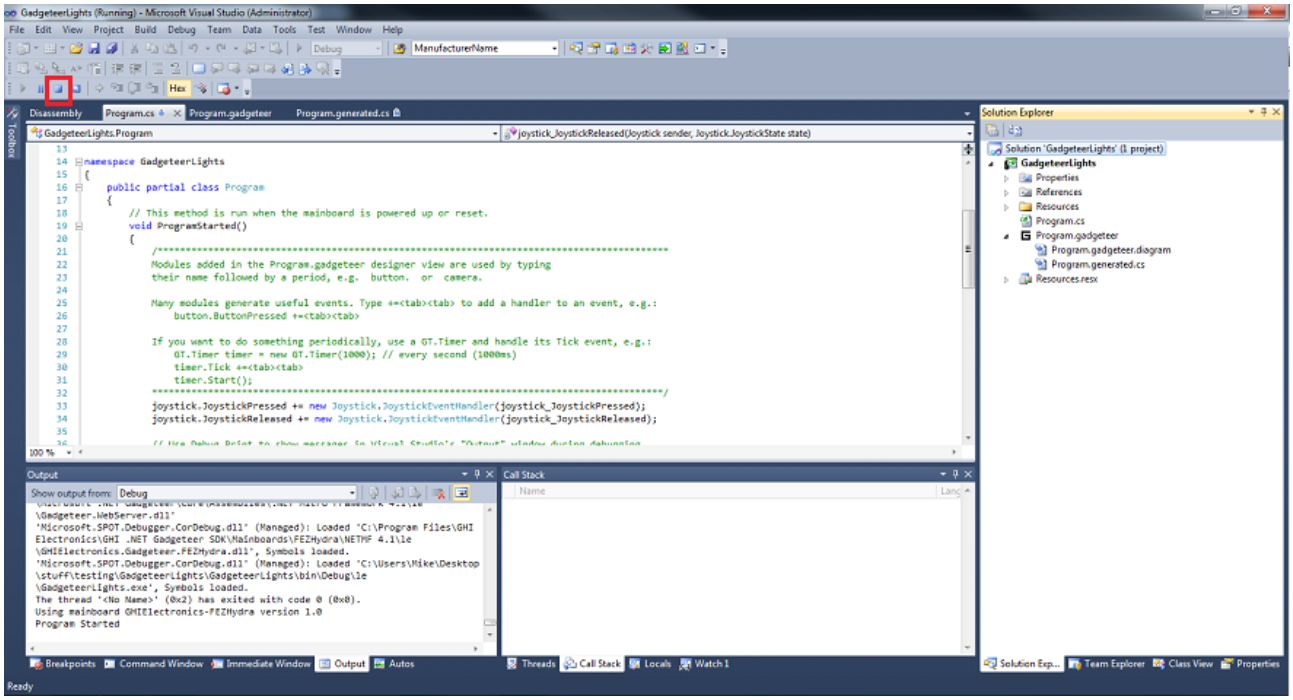
Make sure that the **Output Window** is visible by pressing the **CTRL + ALT + O** key combination on your keyboard. If you have enabled sounds, you should hear Windows make the "USB disconnected" sound, followed by the "USB connected" sound as the Mainboard reboots. The **Output Window** should show 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 start pressing the **Joystick** button. Cover the **LightSense** module, and push the **Joystick** button. The current light percentage will be displayed in the output window where you saw "Program Started". 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 an Interactive Light Show

An interesting extension of the light application in the previous example is programming the application to change the lights based on the Joystick position 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 100 milliseconds (0.1 seconds).

```
GT.Timer timer = new GT.Timer(100);
```

Create the delegate to handle the GT.Timer.Tick event, and start the timer. The following code shows the set-up in the ProgramStarted method.

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

The implementation of the GT.Timer.Tick event is shown in the following example. This will read the Joystick position, and turn the appropriate light on the **LED7R** on.

```
void timer_Tick(GT.Timer timer)
{
    double x = joystick.GetJoystickPostion().X;
    double y = joystick.GetJoystickPostion().Y;

    if (y >= 0.6)
    {
        if (x >= 0.65)
        {
            lED7R.TurnLightOn(2, true);
        }
        else if (x <= 0.35)
        {
            lED7R.TurnLightOn(6, true);
        }
        else
        {
            lED7R.TurnLightOn(1, true);
        }
    }
    else
    {
        if (x >= 0.65)
        {
            lED7R.TurnLightOn(3, true);
        }
        else if (x <= 0.35)
        {
            lED7R.TurnLightOn(5, true);
        }
        else
        {
            lED7R.TurnLightOn(4, true);
        }
    }
```

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

```
using System;
using System.Threading;
using Microsoft.SPOT;
using Microsoft.SPOT.Presentation;
using Microsoft.SPOT.Presentation.Controls;
using Microsoft.SPOT.Presentation.Media;
using Microsoft.SPOT.Touch;

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

namespace GadgeteerLights
{
    public partial class Program
    {
        GT.Timer timer = new GT.Timer(100);

        // This method is run when the mainboard is powered up or reset.
        void ProgramStarted()
        {
            /*******************************************************************************************
            Modules added in the Program.gadgeteer designer view are used by typing
            their name followed by a period, e.g.  button.  or  camera.

            Many modules generate useful events. Type +=<tab><tab> to add a handler to an event, e.g.:
                button.ButtonPressed +=<tab><tab>

            If you want to do something periodically, use a GT.Timer and handle its Tick event, e.g.:
                GT.Timer timer = new GT.Timer(1000); // every second (1000ms)
                timer.Tick +=<tab><tab>
                timer.Start();
            *******************************************************************************************/
            joystick.JoystickPressed += new Joystick.JoystickEventHandler(joystick_JoystickPressed);
            joystick.JoystickReleased += new Joystick.JoystickEventHandler(joystick_JoystickReleased);

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

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

        void joystick_JoystickReleased(Joystick sender, Joystick.JoystickState state)
        {
            Debug.Print("Joystick Released");
            led7R.TurnLightOn(7, true);
        }

        void joystick_JoystickPressed(Joystick sender, Joystick.JoystickState state)
        {
            Debug.Print("Joystick Pressed");

            Debug.Print("Light Sensor %: " + lightSensor.ReadLightSensorPercentage());
            led7R.TurnLightOff(7);
        }

        void timer_Tick(GT.Timer timer)
        {
            double x = joystick.GetJoystickPostion().X;
            double y = joystick.GetJoystickPostion().Y;

            if (y >= 0.6)
            {
                if (x >= 0.65)
                {
                    led7R.TurnLightOn(2, true);
                }
                else if (x <= 0.35)
                {
                    led7R.TurnLightOn(6, true);
                }
                else
                {
                    led7R.TurnLightOn(1, true);
                }
            }
            else
            {
                if (x >= 0.65)
                {
                    led7R.TurnLightOn(3, true);
                }
                else if (x <= 0.35)
                {
                    led7R.TurnLightOn(5, true);
                }
                else
                {
                    led7R.TurnLightOn(4, true);
                }
            }
        }
    }
}
```

Troubleshooting

This section describes common issues that you may encounter and provides suggestions on 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 MFDeploy 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 USBClientSP 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 initializion 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, recconnect 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.