.NET C# for IoT and Robotics

.NET C# developers who code phones and PCs can as well code circuits and robotics. This guide goes hand-in-hand with GHI Electronics' SITCore Experimenter Kit as a step-by-step instructional tutorial.

Copyright © 2021 GHI Electronics, LLC

www.GHIElectronics.com

By: Gus Issa

TABLE OF CONTENTS

Introduction 5
- Prerequisite 5

SITCore Experimenter Kit 6
- SITCore FEZ Bit 6
- BrainBot 8
- BrainClip 9

TinyCLR OS 11

SITCore 12

Getting Started 13
- Firmware Update 13
- Slow Clock Option 13
- Boot-up Options 14
- Visual Studio Setup 15
- Blinky 15
- Top Level Statements 17
- Debugging 17

Digital Pins 18
- Digital Outputs 18
- Digital Inputs 20
- Digital Input Events 24

PWM 27
- Energy Levels 27
- Sounds 28
- Software PWM 30
- Servo Motors 30

Analog Input & Output 33
- Analog Inputs (ADC) 33
- Analog Outputs (DAC) 37

Serial Interfaces 38
- UART 38
  - Events 39
  - RS232 & RS485 40
  - Terminal Software 41

SPI 42
I2C 44
CAN 46
Digital Signals 48
Addressable LEDs 48
Ultrasonic Sensor 51
IR Remote Control 53
Loading Resources 56
Displays 58
Basic Graphics 58
Native Graphics 60
Images 61
Fonts 62
Artificial Intelligence 63
Networking 66
WiFi Setup 66
Sockets 68
UDP 68
TCP 70
HTTP 71
Telnet 74
TLS 76
MQTT 77
Cloud Services 78
Adafruit IO 78
Microsoft Azure 80
Other Cloud Services 83
Cryptography 84
XTEA 84
RSA 84
File System 86
SD Cards 86
USB Mass Storage 87
File System Considerations 88
Time Services 89

Real Time Clock 89
Timers 91
USB Client 92
USB Host 94
Securing IoT 97
Secure Storage 97
IP Protection 98
Data Security 98
Thinking Small 99
Memory Utilization 99
Object Allocation 99
FEZ Bit Reference 100
BrainBot Reference 101
Assembly 101
Pinout 101
BrainClip Reference 102
Digital Modules 102
Analog Modules 102
PWM Modules 102
Special Digital Signals 102
What’s Next? 103

INTRODUCTION

Have you ever thought of some great idea for a product, but you couldn't bring it to life because you did not know hardware? Consider this scenario: You want to make a pocket-GPS-data-logger that saves position, acceleration, and temperature on a memory card. You also want to display some info on a small display.

GPS devices send position data over serial port, so you can easily write some code on the PC to read the GPS data and save it on a file. However, a PC wouldn't fit in your pocket! Another problem is how would you measure temperature and acceleration on a PC? Looks like a PC would not do!

Using a traditional microcontroller is possible but a deep hardware experience is necessary, not a thing that a .NET C# developer is usually familiar with. And even if you did, it takes a very long time to figure out how to accomplish simple tasks, like writing a file to an SD card.

GHI Electronics SITCore family offerings resolve earlier concerns by providing small circuit options that can be coded using C#, thanks to the included TinyCLR OS.

This book will show you how .NET C# can be used with physical computing to control circuits and robots, utilizing the GHI Electronics SITCore Experimenter Kit.

PREREQUISITE

This guide is written for C# developers that want to see how they can program circuits in C# using their existing .NET and Visual Studio experience. It assumes the reader has no hardware knowledge but also assumes the user is comfortable in using Microsoft Visual Studio to build C# applications.

SITCORE EXPERIMENTER KIT

The SITCore Experimenter Kit contains a SITCore Single Board Computer (SBC) called FEZ Bit, BrainClip, and BrainBot. It is an all-in-one complete kit that lets the user see hardware from different angles.

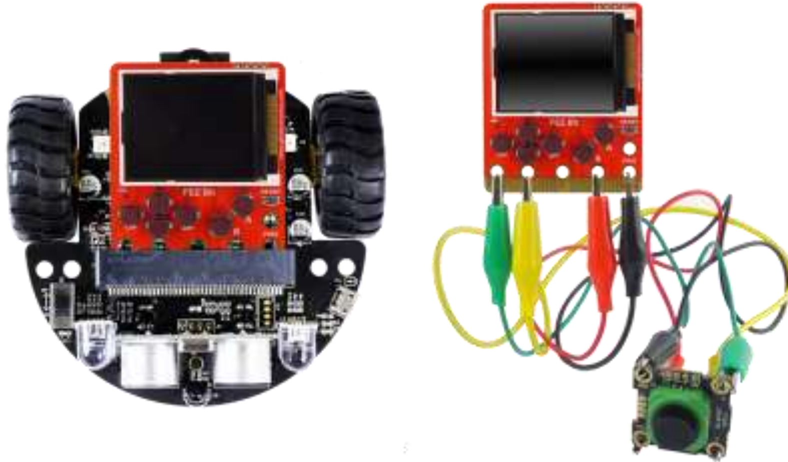
Below are the kit content briefs and there is a reference for each at the end of this guide.

SITCORE FEZ BIT

This micro-computer is the "brain" of the projects we will build. This board runs TinyCLR OS, which is the micro-operating-system that lives on the SITCore board allowing it to understand and run C# code.

Microsoft Visual Studio will be used to deploy our C# application to the hardware, the FEZ Bit. Deployed applications can be debugged through stepping, variable inspection, and breakpoints.

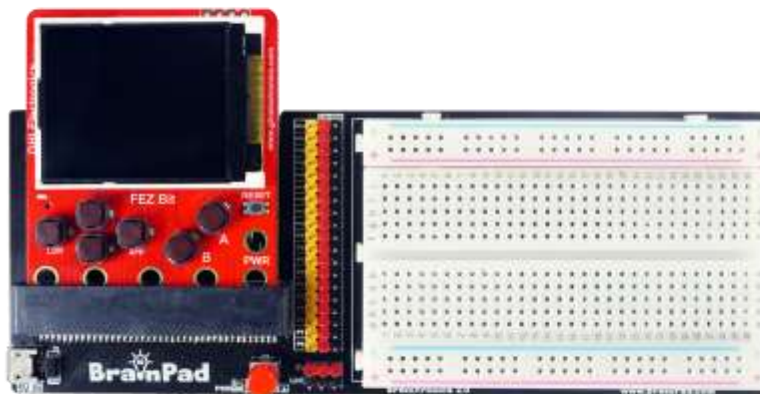
There are two ways to connect circuits to the FEZ Bit: either by plugging it directly into an accessory, such as BrainBot, or by connecting alligator clip wires, such as when using BrainClip modules.



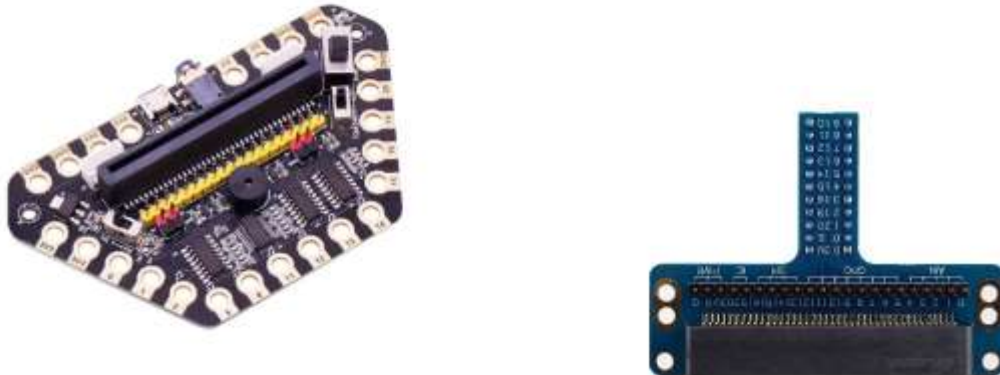
FEZ Bit Key Features:

- Color 160x128 Display.
- 6x User Buttons.
- Buzzer Speaker.
- Micro SD Connector.
- WiFi.

There are breakout boards on the market that brings all available IOs to easily accessible pins. BrainPad BrainTronics is one of the options.



Or search the web for Micro:bit breakout to see the many options available.



BRAINBOT

This little robot has a lithium battery that can keep it going for hours. The FEZ Bit plugs directly into the BrainBot using the edge connector.

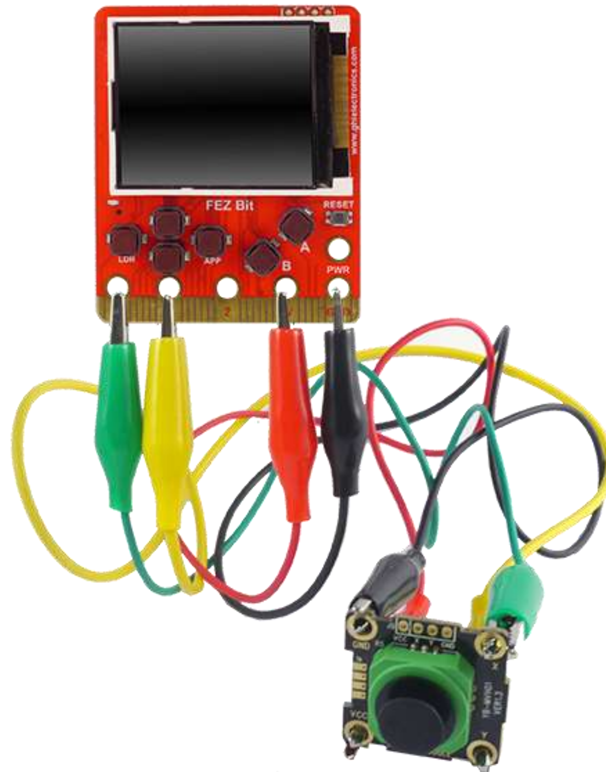
The robot includes multiple sensors and LEDs and only requires minimal assembly.

BrainBot Key Features:

- Line detector for following lines and detecting boundaries.
- Distance sensor for object detection.
- Color headlight LEDs.
- 2x individually controlled body LEDs.
- Buzzer for generating sounds.
- Infrared receiver for the remote control.

BRAINCLIP

The BrainClip set includes multiple circuit modules that connect to the FEZ Bit using the included alligator clip wires.



The circuit modules include a wide variety that help in demonstrating the use of multiple technologies, such as GPIO, ADC, PMW, signal decoders and more.

The set includes:

- Color LED light
- Color LED ring with 8x individually controllable lights
- Distance sensor
- Buzzer speaker
- Red button
- Green button
- Sound sensor
- Analog rocker module
- Motion detector
- Infrared receiver
- Infrared remote control (battery included)
- 10x alligator clip wires
- Reusable case

TINYCLR OS

GHI Electronics TinyCLR OS is micro-operating-system that runs on GHI Electronics SITCore family of products - giving them a long list of supported features, and all coded in C#, right from Microsoft Visual Studio!

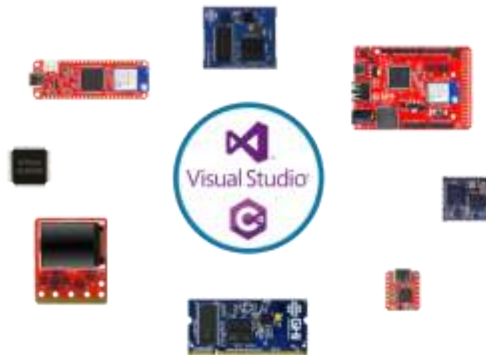
See the TinyCLR OS features page https://www.ghielectronics.com/tinyclr/features/ .

CORE FEATURES	WORLD CLASS SECURITY	CLOUD & NETWORKING
Productivity Enhancements • Debugging • Memory Management • Unmanaged Heap • NuGet based libraries • Mass production tools • Device Info Modern OS Features • Multithreading • Reflection • Collections • Serialization • Encoding & Decoding • Application Domain • Marshal • Regex	Application Security • IP protection • Encrypted in-field update • Secure storage area • Secure boot Data Security • TLS 1.3 • Cryptography • Hashing	Networking • Networking Core • HTTP/HTTPS • MQTT • PPP • TLS 1.3 Interfaces • WiFi • Ethernet • Cellular Cloud support • Azure, AWS, Google • IFTTT, Adafruit IO
HARDWARE CONTROL	MULTIMEDIA READY	ADDITIONAL FEATURES
Pin Level Control • GPIO • PWM • Analog in/out • Signal Control Data Buses • SPI • I2C • UART • USB • CAN • 1-Wire	• Versatile display options • Touch screen support • Powerful UI framework • Native graphics • Image decoders • Font support • Camera interface • MJPEG video • Audio playback	• USB PC Communication • SQLite database • File system • SD/MMC cards • Modbus • Power management • Watchdog timer • Real time clock • Timers • Resources

We will refer to TinyCLR OS as TinyCLR throughout this book for simplicity.

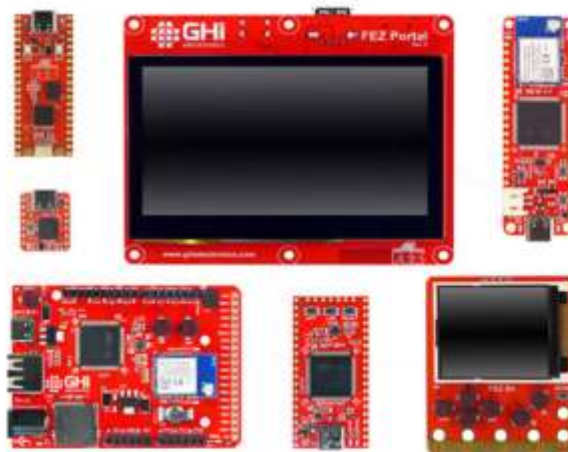
SITCORE

SITCore stands for Secure IoT Core. These are small circuits that are made to be the core of product designs.



The SITCore family consists of multiple options: chipsets, modules, SBCs and development boards. The SBC (Single Board Computer) options are designed to build projects quickly and easily.

Options range from larger boards with touch color displays to very small ones, the size of the tip of your finger!



This book uses FEZ Bit, but other boards work similarly.

GETTING STARTED

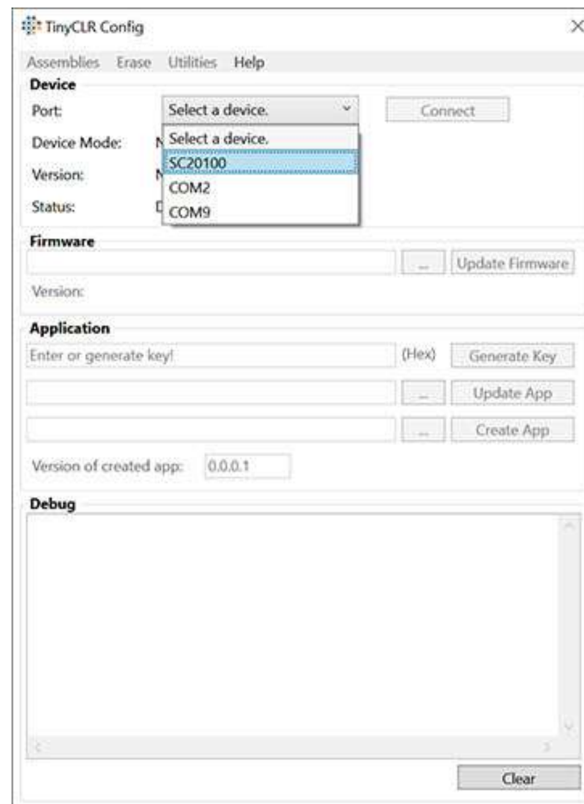
In this chapter, we will blink the on-board LED on FEZ Bit to make sure the development environment and the FEZ Bit are ready and configured properly.

FIRMWARE UPDATE

The first step is loading the FEZ Bit board with appropriate firmware. We recommend using the latest production firmware. The production ready firmware is marked RTW (Ready to Wear). You need to download that firmware from the download page on the docs website. Note that the FEZ Bit uses SC20100 chipset and, therefore, we need the SC20xxx firmware.

https://docs.ghielectronics.com/software/tinyclr/downloads.html

You also need the TinyCLR Config tool. Install and run TinyCLR Config and plug the FEZ Bit board into your PC. The PC will automatically load the appropriate drivers. Now open TinyCLR Config and the **Port** drop-down menu should show the detected SITCore device. The device will either be **COMxx (GHI Electronics)** if the device has no firmware, or the SITCore chipset part number, which is SC20100 in this case.

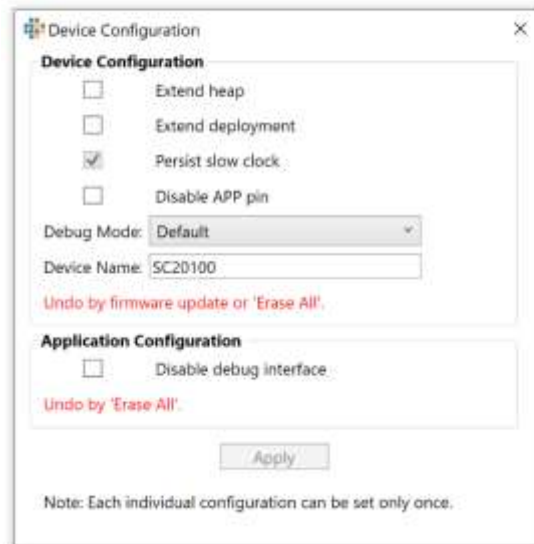


Select the appropriate port and click connect. We now can select the firmware file (downloaded earlier) and click “Update Firmware”.

SLOW CLOCK OPTION

FEZ Bit runs at 480Mhz, which is very fast believe it or not. This is a tiny circuit running 10 times faster than my first computer! Since the FEZ Bit does not have any heat sink or fan, it will run slightly hot. This will not damage the

device, but if you want to run the device cooler, you can set it to run at half the clock speed, which is still 240Mhz fast! To do so, open TinyCLR Config tool, select the device, then push the connect button. Now select **Utilities -> Device Configuration** from the top menu. We now can check the "Persist slow clock" option and click Apply.



The device will now always run at half speed. If you want to go back to full speed, you need to reset the device by doing an erase all or by reloading the firmware.

The Device Configuration window will always show the current settings when it loads up. This will help in determining the current device speed if desired.

BOOT-UP OPTIONS

FEZ Bit checks a couple of the buttons on power up to boot up in a specific state. The APP button (also right button) forces the device to disable the C# application. The application will still be there, but it will simply not run it. Reset or power, cycle the board without holding APP down and the device will run the application.

The LDR button (also left button) is checked on power up and reset to force the device in the loader mode. This is useful if the device needs to be reset completely.

In the unlikely event of the device becoming unresponsive or in an unknown state, hold LDR low and reset the device while holding LDR low. The device will boot up in loader mode. TinyCLR config will see a COMxx (GHI Electronics). Connect to it and then select **Erase -> Erase All** from the top menu.

VISUAL STUDIO SETUP

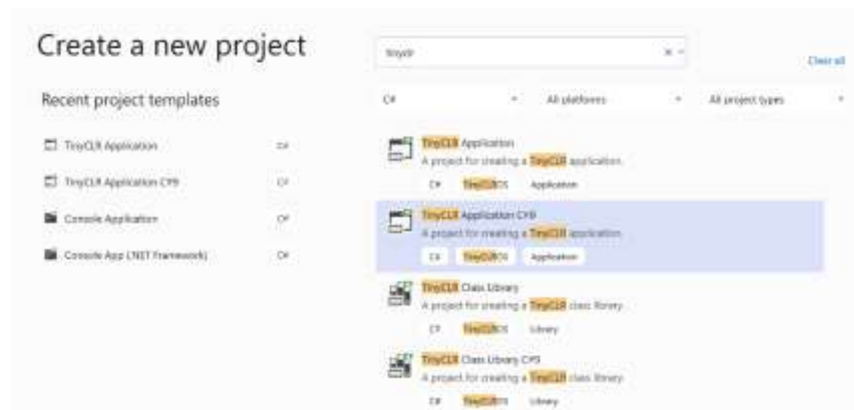
If you do not have it installed already, install Microsoft Visual Studio 2022. Any edition will work, including the free community edition. The ".NET Desktop Development" option is required during installation.

Open Visual Studio and from the top menu select **Extensions -> Manage Extensions** and search for TinyCLR under the **Online > Visual Studio Marketplace**. Download and install **TinyCLR OS Project System**.

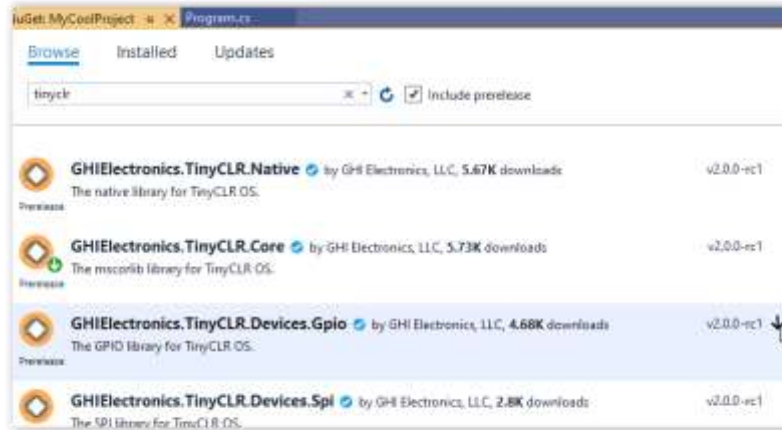
BLINKY

If you are new to hardware, blinking an LED is the "Hello World" of circuits. We will begin by starting Visual Studio and creating a new project, like you would with any other C# project. The only exception is now we need to find the TinyCLR template and start a **TinyCLR Application C#9**.

Note how we have chosen the C#9 option. More on why we did that later.



You can code now just like you would a desktop C# application. Additionally, TinyCLR OS brings new NuGet libraries for hardware access. For example, blinking an LED requires the GPIO (General Purpose Input Output) library. Go to **Manage NuGet Packages…** and search for "tinyclr gpio". Install the **GHIElectronics.TinyCLR.Devices.Gpio** library.



Do the same for **GHIElectronics.TinyCLR.Pins** library and now you are ready to blink! The Pins library includes the available pins, so you do not need the hardware schematics or documentation. For example, FEZ Bit's on-board LED is predefined as FEZBit.GpioPin.Led.

The code first opens the LED pin and then configures the pin to an output. The infinite while loop sets the pin to high and low with some delays in between. We will cover GPIO later, but for now run the code as is.

```
using System;
using System.Diagnostics;
using System.Threading;

using GHIElectronics.TinyCLR.Devices.Gpio;
using GHIElectronics.TinyCLR.Pins;

namespace test {
    class Program {

        static void Main() {
            Debug.WriteLine("Hello World!");
            var led = GpioController.GetDefault().OpenPin(FEZBit.GpioPin.Led);

            led.SetDriveMode(GpioPinDriveMode.Output);

            while(true) {
                led.Write(GpioPinValue.High);
                Thread.Sleep(100);
                led.Write(GpioPinValue.Low);
                Thread.Sleep(300);
            }
        }
    }
}
```

Run the program using F5, like with any other C# program.

Note how the Debug.WriteLine() method sends the string to Visual Studio's output window. By the way, if the FEZ Bit is running without Visual Studio attached, then the string is discarded.

TOP LEVEL STATEMENTS

When we created our project, we selected the C#9 application option. This was done as we will provide examples in this book as a top-level-statement programs. The previous example will look like this in a top-level-statement format.

```
using System;
using System.Diagnostics;
using System.Threading;

using GHIElectronics.TinyCLR.Devices.Gpio;
using GHIElectronics.TinyCLR.Pins;

Debug.WriteLine("Hello World!");
var led = GpioController.GetDefault().OpenPin(FEZBit.GpioPin.Led);

led.SetDriveMode(GpioPinDriveMode.Output);

while (true) {
    led.Write(GpioPinValue.High);
    Thread.Sleep(100);
    led.Write(GpioPinValue.Low);
    Thread.Sleep(300);
}
```

We will use top-level-statements for sample code throughout this book as much as possible as they are simpler.

DEBUGGING

Deploying programs (using F5) from Visual Studio to the FEZ Bit will transfer the compiled program to the FEZ Bit over the USB cable. Visual Studio will then attach the debugger allowing you to pause the program, insert breakpoints and inspect variables.

If the debugger is not needed, you can load the program a bit faster by using ctrl+F5 shortcut.

DIGITAL PINS

Processors usually have many “digital” pins that can be used as inputs or outputs. When saying “digital pins” we mean that the pin can be set to “one” or “zero”, nothing else. A “one” means the pin is active, it is on, it is high, it has voltage on it. When the pin is “zero” it is inactive, it is off, it has no voltage on it.

TinyCLR supports digital input and output pins through **GHIElectronics.TinyCLR.Devices.Gpio** NuGet library and **GHIElectronics.TinyCLR.Devices.Gpio** namespace.

DIGITAL OUTPUTS

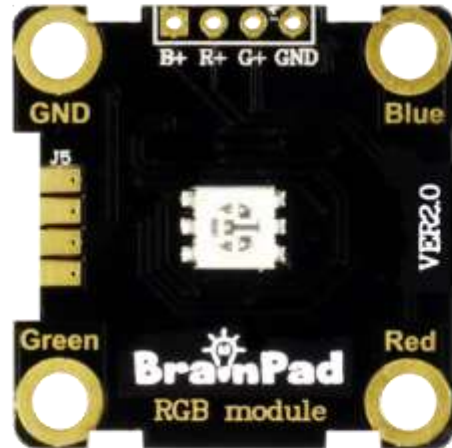
We know that a digital output pin can be set to zero or one. But note that one doesn't mean it is 1 volt. It means that the pin is supplying voltage. If the processor is powered off 3.3V, then the state 1 on a pin means that there is 3.3V on the output pin. By the way, it is not going to be exactly 3.3V but very close. When the pin is set to zero, then its voltage is very close to zero volts.

Note, those digital pins are very weak! They can't be used to drive devices that require a lot of power. For example, a motor may run on 3.3V, but you CANNOT connect it directly to the processor's digital pin. That is because the processor output is 3.3V but with very little power. It is meant to be a “signal” that drives the circuit that, in turn, drives the motor. You can still drive a small LED directly from a pin.

Here is the code used to blink the onboard LED as a digital output.

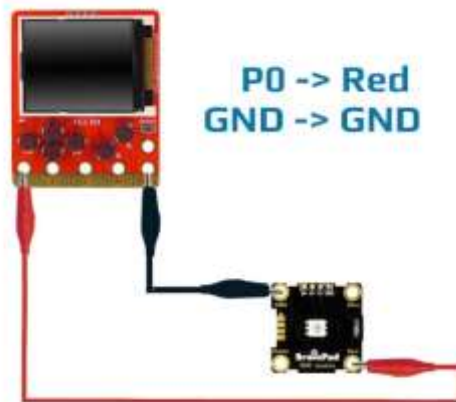
```
using System;
using System.Diagnostics;
using System.Threading;

using GHIElectronics.TinyCLR.Devices.Gpio;
using GHIElectronics.TinyCLR.Pins;


Debug.WriteLine("Hello World!");
var led = GpioController.GetDefault().OpenPin(FEZBit.GpioPin.P0);

led.SetDriveMode(GpioPinDriveMode.Output);

while (true) {

    led.Write(GpioPinValue.High);
    Thread.Sleep(100);
    led.Write(GpioPinValue.Low);
    Thread.Sleep(300);
}
```

Run the program and observe the LED. Things are getting more exciting! How about we now control the BrainClip RGB LED?

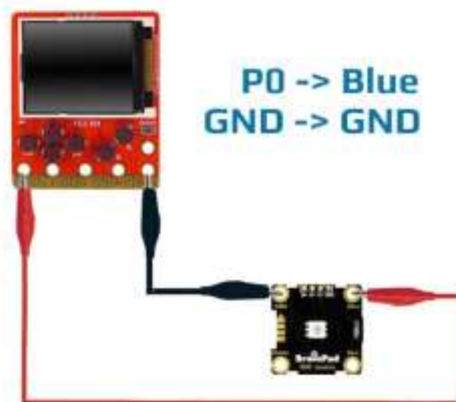


This is an RGB LED, meaning it has three mini light elements inside that are capable of producing Red, Green, and Blue colors. These are the primary colors used to make up any other colors. More on that in a minute.

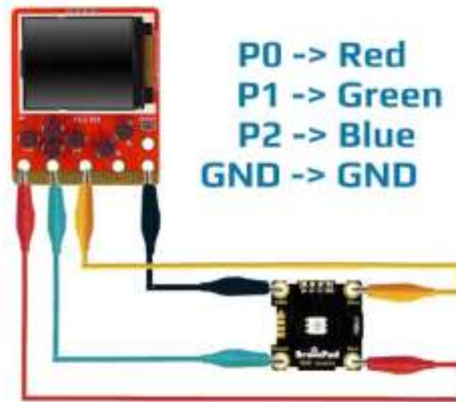
Change the code to control `FEZBit.GpioPin.P0` pad instead of the `FEZBit.GpioPin.Led`. Now, use the alligator clip wires to connect the pad labeled Red on the RGB module to the first pad P0 on the FEZ Bit. Also, connect another alligator clip wire from the GND pad on the FEZ Bit to the GND pad on the RGB module.



Run the program and the RGB module should blink Red, awesome! Move the alligator clip from Red pad to Blue pad on the RGB module and the LED will now blink Blue.



We now want to control all three primary colors. Connect 4 alligator clip wires as shown in this diagram.



You can now set the LED to any of the 8 combinations. Why 8? 2 to the power of 3 is 8! That is 7 colors plus off, which is “Black”.

This example will blink the LED Purple. What 2 colors are used to make up the color Purple?

```
using System;
using System.Diagnostics;
using System.Threading;

using GHIElectronics.TinyCLR.Devices.Gpio;
using GHIElectronics.TinyCLR.Pins;

var Red = GpioController.GetDefault().OpenPin(FEZBit.GpioPin.P0);
var Green = GpioController.GetDefault().OpenPin(FEZBit.GpioPin.P1);
var Blue = GpioController.GetDefault().OpenPin(FEZBit.GpioPin.P2);

Red.SetDriveMode(GpioPinDriveMode.Output);
Green.SetDriveMode(GpioPinDriveMode.Output);
Blue.SetDriveMode(GpioPinDriveMode.Output);

while (true) {

    //purple
    Red.Write(GpioPinValue.High);
    Blue.Write(GpioPinValue.High);
    Thread.Sleep(1000);
    //off
    Red.Write(GpioPinValue.Low);
    Blue.Write(GpioPinValue.Low);
    Thread.Sleep(1000);
}
```

DIGITAL INPUTS

Digital inputs sense if the state on its pin is high or low. There are limitations on those input pins. For example, the minimum voltage on the pin is 0 volts. A negative voltage may damage the pin or the processor. Also, the maximum you can supply to the pin must be less than the processor power source voltage, which is 3.3V on the FEZ Bit. However, the pins on the FEZ Bit are 5V-tolerant. This means that even though the processor runs on 3.3V, the input pins can tolerate up to 5V. But why 5V? Older digital circuits ran on 5V. And many digital circuits today are 5V. Being 5V tolerant allows us to use any of those digital circuits with the FEZ Bit.

Important note: 5V-tolerant doesn't mean the processor can be powered off 5V. Always power it with 3.3V. Only the input pins can tolerate 5V on them.

Input pins are used very similarly to output pins. In fact, all pins are capable of being outputs and inputs. We would only need to set the drive mode of a pin to an input. In the following example, we will blink the LED like we did before, but this time we will add another delay when button A is pressed. Meaning the LED will blink slower while button A is pressed.

```
using System;
using System.Diagnostics;
using System.Threading;

using GHIElectronics.TinyCLR.Devices.Gpio;
using GHIElectronics.TinyCLR.Pins;

var BtnA = GpioController.GetDefault().OpenPin(FEZBit.GpioPin.ButtonA);
BtnA.SetDriveMode(GpioPinDriveMode.InputPullUp);

var LED = GpioController.GetDefault().OpenPin(FEZBit.GpioPin.Led);
LED.SetDriveMode(GpioPinDriveMode.Output);

while (true) {
    if (BtnA.Read() == GpioPinValue.Low)
        Thread.Sleep(500);

    LED.Write(GpioPinValue.High);
    Thread.Sleep(100);

    LED.Write(GpioPinValue.Low);
    Thread.Sleep(100);
}
```

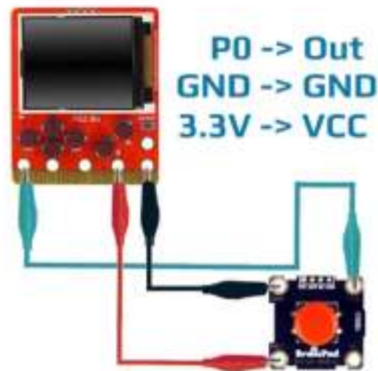
When setting the drive mode of the button, we used InputPullUp, but what does that mean?

First let’s cover how buttons work. A button is a mechanical device that, when pressed, creates an electrical connection between 2 pins. In the case of the FEZ Bit, the 2 pins are connected to a pin on the micro and to ground. So, when the button is not pressed, the pin is not connected to anything and, when the button is pressed, the pin is connected to ground, which is zero.

Unconnected input pins, like when a button is not pressed, are called floating. You would think that unconnected input pins are low with zero volts on them, but this is not true. When a pin is configured as an input and is not connected, it is open for any surrounding noise, which can make the pin high or low. To take care of this issue, modern processors include internal weak pull-down or pull-up resistors that are usually controlled by software. Enabling the pull-up resistor will pull the pin high. Note that the pull-up resistor doesn't make a pin high, but it pulls it high. If nothing is connected, then the pin is high by default.

Can you wire up the BrainClip button modules?

Connect one of the Button Modules to the FEZ bit as shown in the diagram below.



VCC is commonly used in electronics to indicate a voltage source connection. Earlier we discussed pull-up resistors. Well, this button has a pull-up built in it. Meaning you can use Input or InputPullUp and both will work just fine. You can use the same code we used to read button A, except this time change the FEZBit.GpioPin.ButtonA to FEZBit.GpioPin.P0.

```
using System;
using System.Diagnostics;
using System.Threading;

using GHIElectronics.TinyCLR.Devices.Gpio;
using GHIElectronics.TinyCLR.Pins;

var BtnA = GpioController.GetDefault().OpenPin(FEZBit.GpioPin.P0);
BtnA.SetDriveMode(GpioPinDriveMode.InputPullUp);

var LED = GpioController.GetDefault().OpenPin(FEZBit.GpioPin.Led);
LED.SetDriveMode(GpioPinDriveMode.Output);

while (true) {
    if (BtnA.Read() == GpioPinValue.Low)
        Thread.Sleep(500);

    LED.Write(GpioPinValue.High);
    Thread.Sleep(100);

    LED.Write(GpioPinValue.Low);
    Thread.Sleep(100);
}
```

The BrainBot line detector is also a good example of a Digital Input and Digital Output sensor working together. The reflectors on the bottom are used to detect black/white areas under the robot. This can be a black line the robot is following. They are found right next to the caster wheel on the sides of the battery.

The way these sensors work is very simple. There are 2 LED-like elements on the sensor; one is blue and one is black. One of those elements will send a beam of light. The other element will check to see if there is light reflection.

The BrainBot sensors are connected to pins P13 and P14 on the FEZ Bit.

```
using System.Diagnostics;
using System.Threading;
using GHIElectronics.TinyCLR.Devices.Gpio;
using GHIElectronics.TinyCLR.Pins;

var left = GpioController.GetDefault().OpenPin(FEZBit.GpioPin.P13);
left.SetDriveMode(GpioPinDriveMode.Input);
var right = GpioController.GetDefault().OpenPin(FEZBit.GpioPin.P14);
right.SetDriveMode(GpioPinDriveMode.Input);
Debug.WriteLine("Ready...");

while (true) {
    if (left.Read() == GpioPinValue.High)
        Debug.WriteLine("left");
    if (right.Read() == GpioPinValue.High)
        Debug.WriteLine("right");

    Thread.Sleep(100);
}
```

When the robot is sitting on a white (reflective) surface, the output will not show anything. But moving the surface to black/non-reflective surface will start printing the sensor that is not seeing reflection. Note how moving the robot away from any surface will cause no reflection and, therefore, the robot will see if it is black. Use the included tracking map to try out the sensors.

Note, the reflector sensor on the BrainBot will not work properly in direct sunlight, so try it indoors.

DIGITAL INPUT EVENTS

Pooling a pin constantly to check if its status has changed is not the best way to handle input pins. This wastes processor time on something not important. You would be checking the pin, maybe, a million times before it is pressed! Activating an event allows the hardware to call the event only if a pin state has changed.

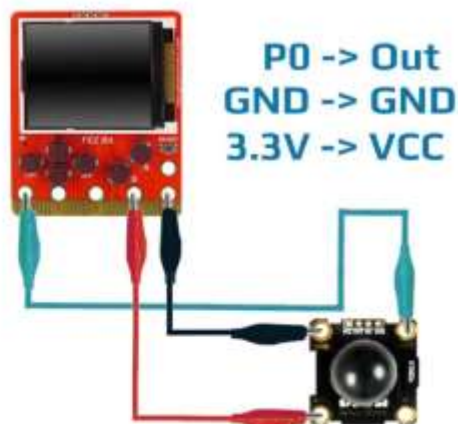
The event can be triggered on FallingEdge (when a pin goes from high to low) or on RisingEdge (when a pin goes from low to high). And of course, we can trigger on both edges, as we have in the following example.

```
using System;
using System.Diagnostics;
using System.Threading;
using GHIElectronics.TinyCLR.Devices.Gpio;
using GHIElectronics.TinyCLR.Pins;

var led = GpioController.GetDefault().OpenPin(FEZBit.GpioPin.Led);
led.SetDriveMode(GpioPinDriveMode.Output);

var button = GpioController.GetDefault().OpenPin(FEZBit.GpioPin.ButtonA);
button.SetDriveMode(GpioPinDriveMode.InputPullUp);
button.ValueChangedEdge = GpioPinEdge.FallingEdge | GpioPinEdge.RisingEdge;
button.ValueChanged += Button_ValueChanged;
Thread.Sleep(Timeout.Infinite);
void Button_ValueChanged(GpioPin sender, GpioPinValueChangedEventArgs e) {
    if (e.Edge == GpioPinEdge.FallingEdge) {
        led.Write(GpioPinValue.High);
    }
    else {
        led.Write(GpioPinValue.Low);
    }
}
```

The previous example used an event that is triggered on both Rising and Falling Edges. The event would then set the LED to turn on when the button is pressed and shut off when the button is released.

How about we now build an alarm system? We will use the motion sensor in this next example.

Since we do not know how to generate sounds yet, we will simply blink the LED when we detect motion. The motion sensor OUT pin goes high when it detects motion. Connect the FEZ Bit to the Motion Sensor as shown in the diagram below.




```
using System;
using System.Diagnostics;
using System.Threading;
using GHIElectronics.TinyCLR.Devices.Gpio;
using GHIElectronics.TinyCLR.Pins;

var led = GpioController.GetDefault().OpenPin(FEZBit.GpioPin.Led);
led.SetDriveMode(GpioPinDriveMode.Output);

Thread blinkT = null;
void Blinker() {
    while (true) {
        led.Write(GpioPinValue.High);
        Thread.Sleep(300);
        led.Write(GpioPinValue.Low);
        Thread.Sleep(300);
    }
}
```

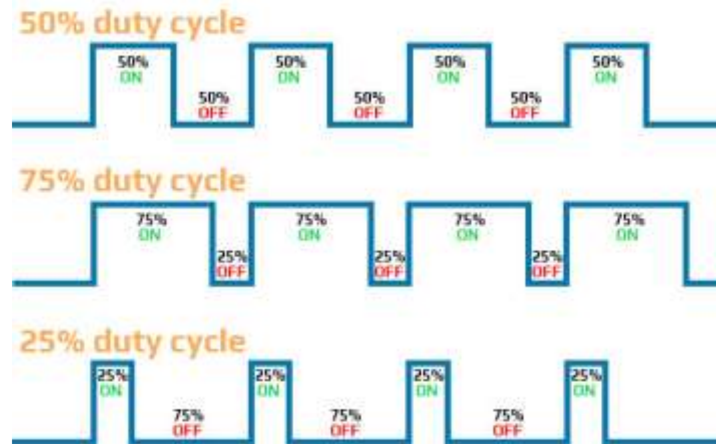
```
var motion = GpioController.GetDefault().OpenPin(FEZBit.GpioPin.P0);
motion.SetDriveMode(GpioPinDriveMode.Input);
motion.ValueChangedEdge = GpioPinEdge.RisingEdge;
motion.ValueChanged += motion_ValueChanged;
Thread.Sleep(Timeout.Infinite);

void motion_ValueChanged(GpioPin sender, GpioPinValueChangedEventArgs e) {
    if (blinkT == null) {
        blinkT = new Thread(Blinker);
        blinkT.Start();
    }
}
```

The program does not have a mechanism to stop the “alarm”. Meaning once the LED starts blinking it will not stop. You can use the reset button to reset the program and stop the blinking. Then any detected motion by the sensor will trigger the LED into blinking again.

PWM

Pulse Width Modulation (PWM) is very useful and used in many ways. Most systems, including FEZ Bit, include PWM functionality, which is also natively supported by TinyCLR.

When activated, a PWM pin generates a square wave output where you can set the pin's frequency and duty cycle. Frequency is used to set how many times a second a pin will turn on and off. The duty cycle is used to set the ratio of the pin's on state or the pin's off state.



The system has multiple PWM controllers, and each controller has multiple channels. The controller is what generates the frequency, and the channel is what sets the duty cycle. Note how there are multiple channels that share the controller, and so these pins share the same frequency. If you need different frequencies on different pins, then these pins must have different PWM controllers. This example sets the LED to 10Khz with 10% duty cycle.

You'll need to import the GHIElectronics.TinyCLR.Devices.Pwm NuGet/namespace just like we did for pins.

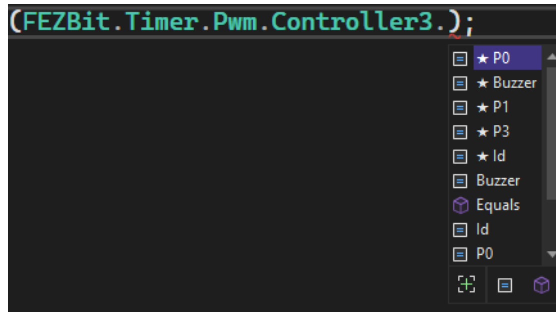
```
using GHIElectronics.TinyCLR.Devices.Pwm;
using GHIElectronics.TinyCLR.Pins;

var controller = PwmController.FromName(FEZBit.Timer.Pwm.Controller1.Id);
var pwmPin = controller.OpenChannel(FEZBit.Timer.Pwm.Controller1.Led);
controller.SetDesiredFrequency(10000);
pwmPin.SetActiveDutyCyclePercentage(0.1);
pwmPin.Start();
```

All three large pads on FEZ Bit support PWM. P0 & P1 use Controller3, while P2 uses Controller5. The onboard LED is on Controller1.

ENERGY LEVELS

Controlling the brightness of an LED or a light bulb is not done by reducing the voltage. Instead, the power/energy level is controlled by using PWM duty cycle. Consider this, an LED is turning on and off 1000 times per second (1000Hz) with 10% duty cycle. You would not see the LED flickering at 1000 times per second. Instead, you will see the average result of the LED being active only half the time. In other words, the LED will be dimmer.

The frequency here is not very important, as the LED is blinking faster than what our eyes can see. The onboard LED on FEZ Bit is connected to a PWM capable pin. To determine what controller it is on, we can use Intellisense to see what channels are available under a specific controller.



We'll need to use Controller1 to access the onboard LED. Now our program will fade the LED in and out indefinitely.

```
using System.Threading;
using GHIElectronics.TinyCLR.Pins;
using GHIElectronics.TinyCLR.Devices.Pwm;

var controller = PwmController.FromName(FEZBit.Timer.Pwm.Controller1.Id);
var led = controller.OpenChannel(FEZBit.Timer.Pwm.Controller1.Led);
controller.SetDesiredFrequency(10000);

double duty = 0.5, speed = 0.01;

led.Start();

while (true) {
    if (duty <= 0 || duty >= 1.0) {
        speed *= -1; //Reverse direction.
        duty += speed;
    }

    led.SetActiveDutyCyclePercentage(duty);
    duty += speed;

    Thread.Sleep(10);
}
```

Remember how we could only set the RGB LED to 7 colors? Well, with PWM, you can set the LED to any color, with millions of options! You can use PWM to set the intensity of each of the RGB color elements to create any color combo.

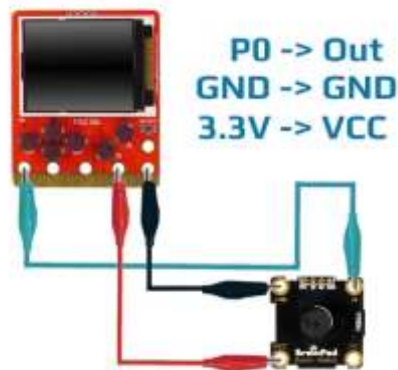
SOUNDS

Generating sounds can also be done using PWM. In this case, frequency is very important, but duty cycle has little effect. FEZ Bit has a buzzer speaker, so let's use it.

```
using System.Threading;
using GHIElectronics.TinyCLR.Pins;
using GHIElectronics.TinyCLR.Devices.Pwm;

var controller = PwmController.FromName(FEZBit.Timer.Pwm.Controller3.Id);
var buzzer = controller.OpenChannel(FEZBit.Timer.Pwm.Controller3.Buzzer);
controller.SetDesiredFrequency(10000);
buzzer.SetActiveDutyCyclePercentage(0.5);

while (true) {
    buzzer.Start();
    for (var f = 500; f < 5000; f += 100) {
```

```
            controller.SetDesiredFrequency(f);
            Thread.Sleep(10);
        }
        buzzer.Stop();
        Thread.Sleep(1000);
    }
```

The code above can be modified to control the buzzer module.



P0 pad is PWM capable, found on Controller3. By the way, you can also connect the PWM pin to headphones or a computer speaker using the 3.5mm connector. Use the alligator clip wires to clip P0 on the top sleeve and GND on the last sleeve.

SOFTWARE PWM

PWM is generated by special hardware inside the micro that runs without the need for any software processing. The software only configures the frequency and duty cycle, but then the pin will generate PWM signal without any further need for software. On the bright side, this means that the signal is very accurate and stable no matter how busy the system might be. Unfortunately, PWM is only available on specific pins.

TinyCLR adds software PWM feature that allows you to generate PWM signals on any pin. This is a great feature, but keep in mind that the system internally needs to use resources to generate the signal. Also, the generated signal will not be 100% accurate. Software PWM is more accurate at lower frequencies, making it a good option for driving servo motors and dimming LEDs at a lower frequency.

To use the software PWM, use the Pwm.Software.Id controller. And then for channel, just use the desired GPIO pin. Note here how the channel is not really a PWM channel like we did before, but a GPIO pin. And this is what the software PWM controller is designed to do.

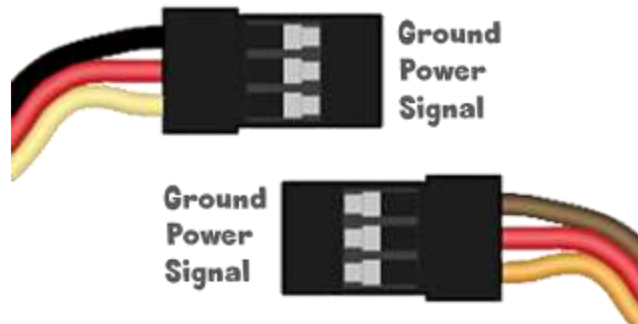
```
using GHIElectronics.TinyCLR.Devices.Pwm;
using GHIElectronics.TinyCLR.Pins;

var softwarePwmController = PwmController.FromName(SC20100.Timer.Pwm.Software.Id);
var pwmPin = softwarePwmController.OpenChannel(FEZBit.GpioPin.P0);

softwarePwmController.SetDesiredFrequency(1000); // set frequency 1KHz
pwmPin.SetActiveDutyCyclePercentage(0.1);
pwmPin.Start();
```

SERVO MOTORS

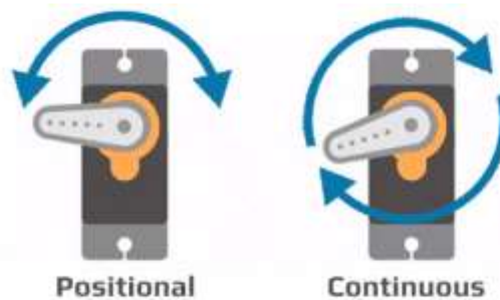
Traditional motors convert electrical energy to mechanical rotation. When it comes to speed, the higher the voltage, the faster the motor. Servo motors are a bit smarter! They take a voltage (power) for energy but also take a signal that comes into the servo to tell it what we want it to do. This page covers the RC Servo motor, originally made for RC (radio controlled) hobby vehicles. On RC Servos, the signal coming in is a pulse that repeats 50 times per second, that is every 20 milliseconds. The incoming pulse size can vary between 1 and 2 milliseconds typically and then repeats every 20 milliseconds. The smart circuit inside the servo sees the pulse and then controls the internal motor based on the pulse size.

The three wires on a servo are usually Black, Red, and Yellow. The Red and Black ones are for power, where Red is positive (power source) and Black is negative (ground). The Yellow wire is the signal. Some other servos have Orange, Red, and Brown. Where Red and Brown are the power and Orange is the signal. If in doubt, always

remember that the middle pin is always the power source, and then the negative pin (the ground) is the darker color, like Brown or Black.



There are two types of servo motors, positional and continuous.



As name suggests, continuous servos keep on rotating, and the positional servos move to a specific position. For example, a positional servo is used to steer the wheels on a car robot or to turn the camera to a specific direction.

Servo motors generally need 5V to operate. You can use the PWR pad on the FEZ Bit, just above the GND pad, to draw 5V directly from the USB cable. This is good for powering 1 servo motor. If you need more motors for a robot or something more complex, a dedicated power source is necessary. To wire a servo to the FEZ Bit we'll need to use pin to pin wires to use inside the servo's plug. Wire as shown in the diagram.

PWM works well with servo motors. But to make things a bit easier, TinyCLR includes a servo motor driver through GHIElectronics.TinyCLR.Drivers.Motor.Servo NuGet/namespace.

The servo signal is relatively slow and, therefore, software PWM can handle it just fine. We will use software PWM on P0 pad to control a servo motor, but hardware PWM will work as well.

```
using System.Threading;
using GHIElectronics.TinyCLR.Pins;
using GHIElectronics.TinyCLR.Devices.Pwm;
using GHIElectronics.TinyCLR.Drivers.Motor.Servo;

var softwarePwmController = PwmController.FromName(SC20100.Timer.Pwm.Software.Id);
var pwmPinPB3 = softwarePwmController.OpenChannel(FEZBit.GpioPin.P0);
var servo = new ServoController(softwarePwmController, pwmPinPB3);
servo.ConfigureAsPositional(false);

while (true) {
    servo.Set(0); // 0 degree
    Thread.Sleep(2000);
    servo.Set(45.0); // 45 degree
    Thread.Sleep(2000);
    servo.Set(90.0);  // 90 degree
    Thread.Sleep(2000);
    servo.Set(180.0); // 180 degree
}
```

ANALOG INPUT & OUTPUT

Analog pins are usually multiplexed with digital pins. Some of the processor pins can be configured to be digital or analog.

ANALOG INPUTS (ADC)

Digital input pins can only read high and low (one or zero), but analog input pins can read the voltage level. Analog input pins are called ADC for Analog to Digital Converter. There are limitations on voltages that can be applied to analog inputs. For example, the FEZ Bit analog input can read voltages anywhere between zero and 3.3V. When the pins are digital, they are tolerant of 5V; but when the same pin is set to analog only up to 3.3V can be used. This limitation of analog inputs is not a big issue usually because most analog signals are conditioned to work with the analog inputs. A voltage divider or an op-amp circuit can be used to get a fraction of the actual signal to scale. For example, if we want to measure the battery voltage that is 6V, then we need to divide the voltage in half using a voltage divider, so the analog pin will only see half the voltage, that is 3V max. In software, we know we have the voltage divided in half, so any voltage we see will need to be multiplied by 2 to give us the actual voltage we are trying to measure.

We will use the light sensor module with our first analog input example. This module outputs an analog value reflecting the ambient light level.

The connections are simple and just like before with other modules.

There are 2 options to read an ADC: by reading the raw value directly or by reading the ratio. Using the ratio is typically always better because you can simply multiply the ratio with your input value before scaling to give you a final useful result.

The ADC libraries are found in the GHIElectronics.TinyCLR.Devices.Adc NuGet/namespace.

```
using System.Diagnostics;
using System.Threading;
using GHIElectronics.TinyCLR.Pins;
using GHIElectronics.TinyCLR.Devices.Adc;

var adc = AdcController.FromName(FEZBit.Adc.Controller3.Id);
var analog = adc.OpenChannel(FEZBit.Adc.Controller3.P0);

while (true) {
    var d = analog.ReadRatio() * 1000;
    Debug.WriteLine("> " + d.ToString("N0"));
    Thread.Sleep(100);
}
```

Can we use ratio to scale the value to 5000 and use that to generate sound on the buzzer? We already know how to use the buzzer, right? It was explained in the PWM section previously.

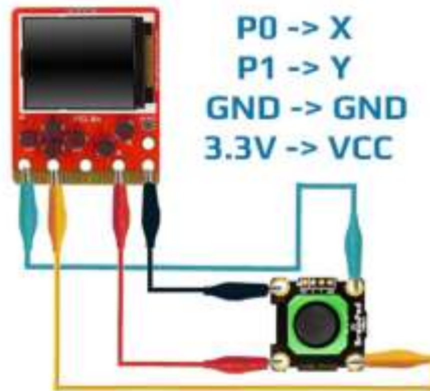
```
using System.Threading;
using GHIElectronics.TinyCLR.Pins;
using GHIElectronics.TinyCLR.Devices.Pwm;
using GHIElectronics.TinyCLR.Devices.Adc;

var adc = AdcController.FromName(FEZBit.Adc.Controller3.Id);
var analog = adc.OpenChannel(FEZBit.Adc.Controller3.P0);

var buzzerController = PwmController.FromName(FEZBit.Timer.Pwm.Controller3.Id);
var buzzer = buzzerController.OpenChannel(FEZBit.Timer.Pwm.Controller3.Buzzer);
buzzerController.SetDesiredFrequency(10000);
buzzer.SetActiveDutyCyclePercentage(0.5);
buzzer.Start();

while (true) {
    var d = analog.ReadRatio() * 5000;
    buzzerController.SetDesiredFrequency(d);
    Thread.Sleep(100);
}
```

Cover the light sensor with your hand to stop it from seeing any light and the sound will change to a higher frequency. Now give the light sensor more light and you will start hearing lower frequencies.

Can you now experiment with returned values and turn a light (LED) on when the environment is dark? You can use the onboard LED or connect the RGB LED module.

With the previous example, we can use the rocker module to change the frequency.

For connections, the module needs power and ground like always, but then it has 2 outputs, X and Y. You can connect these to any of the analog inputs.



Using the same code as the light sensor, we can use the rocker to change the frequency from 0 to 5000.

Let's make things more interesting! We will change the code to sweep through frequencies but use Y to set the top frequency (max value) and the X to control the sweep speed (the increment).

```
using System.Threading;
using GHIElectronics.TinyCLR.Pins;
using GHIElectronics.TinyCLR.Devices.Pwm;
using GHIElectronics.TinyCLR.Devices.Adc;

var adc3 = AdcController.FromName(FEZBit.Adc.Controller3.Id);
var analogX = adc3.OpenChannel(FEZBit.Adc.Controller3.P0);
var adc1 = AdcController.FromName(FEZBit.Adc.Controller1.Id);
var analogY = adc1.OpenChannel(FEZBit.Adc.Controller1.P1);

var buzzerController = PwmController.FromName(FEZBit.Timer.Pwm.Controller3.Id);
var buzzer = buzzerController.OpenChannel(FEZBit.Timer.Pwm.Controller3.Buzzer);
buzzerController.SetDesiredFrequency(10000);
buzzer.SetActiveDutyCyclePercentage(0.5);
buzzer.Start();

while (true) {
    for (var f = 500.0;
        f < analogY.ReadRatio() * 5000;
        f += analogX.ReadRatio() * 100) {
        buzzerController.SetDesiredFrequency(f);
    }
    Thread.Sleep(10);
}
```

The Sound sensor is another analog module we can use to detect sound levels.

Let’s use the sound level to control the brightness of the on board LED. We suggest you try this alone since you will be screaming at the FEZ Bit!

Connect the FEZ Bit to the Sound Sensor Module as shown in the diagram below.

```
using System.Threading;
using GHIElectronics.TinyCLR.Pins;
using GHIElectronics.TinyCLR.Devices.Pwm;
using GHIElectronics.TinyCLR.Devices.Adc;

var adc = AdcController.FromName(FEZBit.Adc.Controller3.Id);
var analog = adc.OpenChannel(FEZBit.Adc.Controller3.P0);
var controller = PwmController.FromName(FEZBit.Timer.Pwm.Controller1.Id);
var led = controller.OpenChannel(FEZBit.Timer.Pwm.Controller1.Led);

controller.SetDesiredFrequency(10000);
led.Start();

while (true) {
    var d = analog.ReadRatio();
    led.SetActiveDutyCyclePercentage(d);

    Thread.Sleep(10);
}
```

ANALOG OUTPUTS (DAC)

An analog capable output can be set to a specific voltage, with a maximum of the system's power source. Analog outputs are like digital outputs where they have limits on how much power they can provide. Analog outputs are even weaker than digital outputs. They are only capable of providing a very little signal to drive the power circuit, maybe drive a power amplifier.

There are 2 DAC pins on the FEZ Bit, but they are found on the smaller pads, P4 (PA4) and P13 (PA5). You will need a breakout board to get to these pins. DAC is rarely used, and you will probably never need it. However, here is an example.

The DAC libraries are provided through the GHIElectronics.TinyCLR.Devices.Dac NuGet/namespace.

```
using System.Threading;
using GHIElectronics.TinyCLR.Pins;
using GHIElectronics.TinyCLR.Devices.Dac;

var dac = DacController.GetDefault();
var analog = dac.OpenChannel(SC20100.Dac.PA4);

double d = 0.5;
double dd = 0.01;

while (true)
{
    analog.WriteValue(d);
    d += dd;
    if (d <= 0 || d >= 1)
        dd *= -1;   //Invert
    Thread.Sleep(10);
}
```

SERIAL INTERFACES

There are many serial interfaces available for data transfer between processors/devices. Each interface has its advantages and disadvantages. We will try to cover them in enough detail so you can use them with TinyCLR.

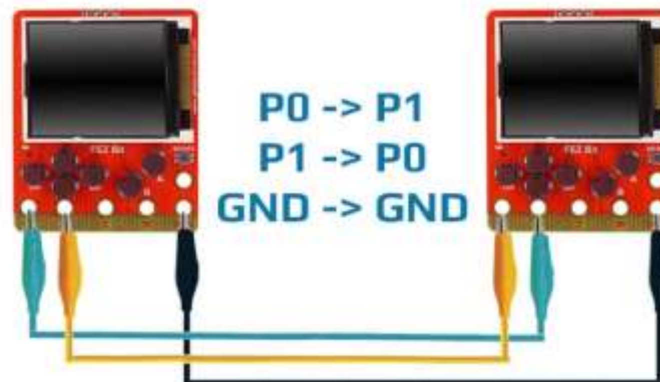
UART

Even though there are many serial interfaces, “serial” commonly refers to UART or RS232. Other busses, like CAN and SPI, still transmit its data serially, but they are not serial ports! On Windows, the serial port is referred to as COM port.

UART is one of the oldest and most common interfaces. Data is sent out on a UART TX pin at a predefined speed, called baudrate. When the transmitter sends out zeros and ones, the receiver is checking for incoming data on RX pin, at the same baudrate. The data is sent one byte at the time.

This covers one direction for data transfer. To transfer the data in the opposite direction, a similar circuit is constructed at the opposite side. Transmit and receive are separate circuits and they can work simultaneously. Each side can send data at any time and can also receive data at any time.

The FEZ Bit exposes UART6 TX on P0 and UART6 RX on P1. You can use this to send data between two FEZ Bit boards. Just keep in mind that RX on one side (receive) needs to go to TX on the other side (transmit). Do not forget to connect GND between the boards!



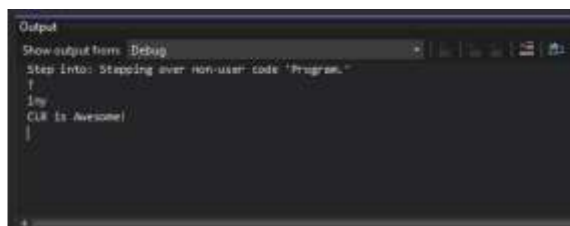
Another easier option is to connect TX to RX directly. This is called loopback. This will result in your board receiving whatever you are sending!

The UART drivers are found in the GHIElectronics.TinyCLR.Devices.Uart NuGet/namespace.

```
using System.Text;
using System.Threading;
using GHIElectronics.TinyCLR.Devices.Uart;
using GHIElectronics.TinyCLR.Pins;
using System.Diagnostics;

var txBuffer = Encoding.UTF8.GetBytes("TinyCLR is Awesome!");
var rxBuffer = new byte[txBuffer.Length];

var myUart = UartController.FromName(FEZBit.UartPort.Uart6);

var uartSetting = new UartSetting() {
    BaudRate = 115200,
    DataBits = 8,
    Parity = UartParity.None,
    StopBits = UartStopBitCount.One,
    Handshaking = UartHandshake.None,
};

myUart.SetActiveSettings(uartSetting);

myUart.Enable();
myUart.Write(txBuffer, 0, txBuffer.Length);

while (true) {
    if (myUart.BytesToRead > 0) {
        var bytesReceived = myUart.Read(rxBuffer, 0, myUart.BytesToRead);
        Debug.WriteLine(Encoding.UTF8.GetString(rxBuffer, 0, bytesReceived));
    }

    Thread.Sleep(20);
}
```

We can see from the code that UART requires other settings beside the baudrate. Those need to be set to match on both ends, just like the baudrate. Note that while you can, in theory, use any number for the baudrate, it is recommended that you use one of the standard values, such as 9600, 19200, 115200.

It is important to realize that UART sends data byte by byte and not in packets. If the sender is sending 100 bytes and the receiver started receiving data somewhere in the middle of the transfer, the receiver will see maybe 36 bytes of the 100 bytes being transmitted. The following reads will see the remaining bytes. This is normal and by design. The receiver needs to handle incoming data with that in mind.



You can find more details at https://en.wikipedia.org/wiki/Universal_asynchronous_receiver-transmitter

EVENTS

Receiving data can be simply done by reading UART and this is the recommended approach when dealing with high levels of UART traffic. However, in cases where data comes slowly, using an event is a better approach.

```
using System.Text;
using System.Threading;
using GHIElectronics.TinyCLR.Devices.Uart;
using GHIElectronics.TinyCLR.Pins;
using System.Diagnostics;

var txBuffer = Encoding.UTF8.GetBytes("TinyCLR is Awesome!");
var rxBuffer = new byte[txBuffer.Length];

var uart = UartController.FromName(FEZBit.UartPort.Uart6);
var uartSetting = new UartSetting() {
    BaudRate = 115200,
    DataBits = 8,
    Parity = UartParity.None,
    StopBits = UartStopBitCount.One,
    Handshaking = UartHandshake.None,
};
uart.SetActiveSettings(uartSetting);
uart.Enable();
uart.DataReceived += uart_DataReceived;
uart.Write(txBuffer, 0, txBuffer.Length);

// do something...
Thread.Sleep(Timeout.Infinite);

void uart_DataReceived(UartController sender, DataReceivedEventArgs e) {
    var bytesReceived = uart.Read(rxBuffer, 0, e.Count);
    Debug.WriteLine(Encoding.UTF8.GetString(rxBuffer, 0, bytesReceived));
}
```

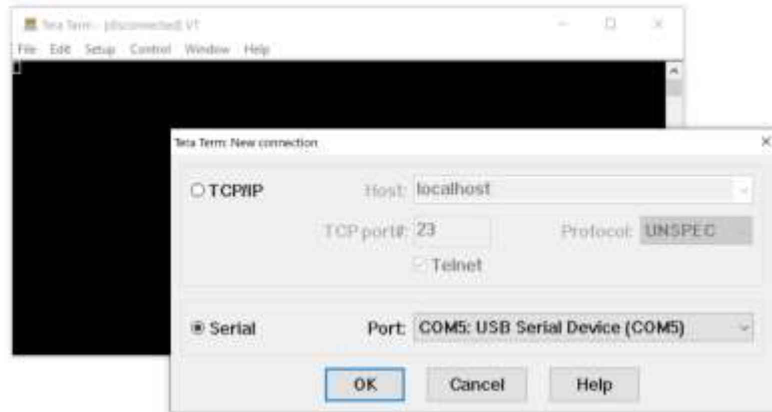
RS232 & RS485

UART connects processor pins directly and it assumes both sides have compatible voltages. UART can also be used in industrial environments. In industrial systems, or when long wires are used, 3.3V or even 5V doesn't provide enough room for error. A circuit is added to change the voltages and how data is treated on the long wires. One of the most common interfaces is called RS232. Computers used to have this before USB, which made it very polar in the past. Another popular interface is RS485, which is immune to noise and, therefore, used in industrial environments.

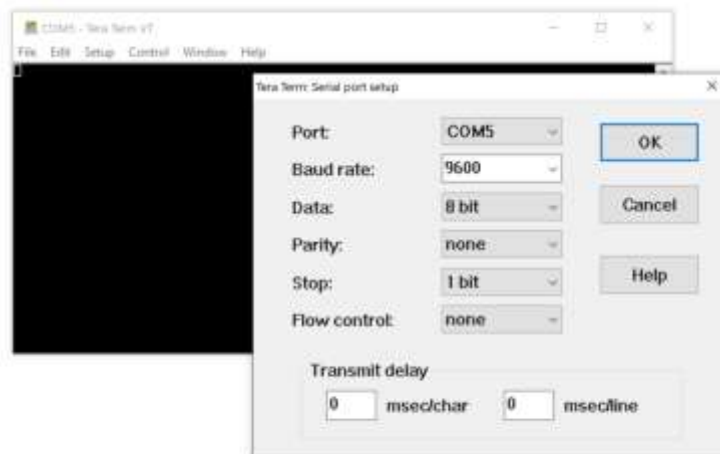
Things have evolved much since then and now many use USB to provide a virtual serial port. This will seem like a serial port to windows, but in fact it is just a USB connection. Even better, some cables, like FTDI TTL-232R-3V3 have UART connections on one side to USB on the other side. There is a small USB<->serial chip that is hidden inside the cable. This cable can connect to FEZ Bit UART on one side and the PC on the other side.

TERMINAL SOFTWARE

Historically, terminal software that run on PCs to send and receive serial (UART/RS232) data were very common. They are still commonly used in embedded systems due to their simplicity and availability. One of the popular terminals is Tera Term.



As you can see from the image, the terminal can use TCP/IP. But for serial, you simply select the COM port you need to use. Note that this COM port can be a virtual serial port. And can even be a USB<->serial cable that connects the PC to FEZ Bit. This allows you to build software on FEZ Bit that prints our messages to the serial port that you will then see on Tera Term. Just do not forget to configure the serial bus to match between FEZ Bit and the Terminal.



SPI

Serial Peripheral Interface (SPI) uses 3 or 4 wires for transferring data. In contrast, UART requires both sides to agree on a predetermined baudrate. This is different on SPI as there is a serial clock (SCK) pin that is used to sync both sides. If you know what a shift register is, SPI is simply a shift register!

There is always a master device, which is the device that generates the clock. And then there are one or more slaves that monitor the clock. The FEZ Bit is always the SPI master. The display found on the FEZ Bit is an SPI slave device.

Data is transferred in one direction using Master Out Slave In (MOSI) pin and in the other direction on the Master In Slave Out (MISO) pin. Some devices may have only one of these pins. For example, the display only needs data coming from the master. Only MOSI pin is needed. The display does not need to send data back ever!

When there are multiple slaves on the SPI bus, a Chip Select (CS) pin is used to instruct the slaves to handle or ignore the next SPI transfer. CS pin is sometimes referred to as Slave Select (SSEL).

There is an SPI bus available on the edge connector of FEZ Bit:

- SCK: P13
- MISO: P14
- MOSI: P15

Any of the other pins can be used as a CS.

The easiest way to handle SPI transactions is by using TransferFullDuplex. This is because it is the closest method to how SPI works, which is swapping bytes between the master and the selected slave. In SPI, you can't read a byte without writing a byte and you can't write a byte without reading a byte. You are always swapping bytes, meaning reading and writing at the same time! Yes, there is a Write method to only write, but this method is reading and discarding the incoming bytes.

The SPI drivers are found in GHIElectronics.TinyCLR.Devices.Spi NuGet/namespace.

```
using GHIElectronics.TinyCLR.Devices.Spi;
using GHIElectronics.TinyCLR.Devices.Gpio;
using GHIElectronics.TinyCLR.Devices.Spi;
using GHIElectronics.TinyCLR.Pins;

var settings = new SpiConnectionSettings() {
    ChipSelectType = SpiChipSelectType.Gpio,
    ChipSelectLine = GpioController.GetDefault().OpenPin(FEZBit.GpioPin.P16),
    Mode = SpiMode.Mode1,
    ClockFrequency = 4_000_000,
};

var controller = SpiController.FromName(FEZBit.SpiBus.Edge);
var device = controller.GetDevice(settings);

device.Write(new byte[] { 1, 2 }); //Write something, discard incoming
device.TransferFullDuplex(...);    //Swap data, send and read simultaneously
```

To put SPI to an actual use, we will use the FEZ Bit display. A driver is provided to make using the display easier, but the source code is fully available.

The display driver is found in GHIElectronics.TinyCLR.Drivers.Sitronix.ST7735 NuGet/namespace.

We will create an InitDisplay() method that you can start adding to your programs to utilize the display.

```
using GHIElectronics.TinyCLR.Drivers.Sitronix.ST7735;
using GHIElectronics.TinyCLR.Devices.Spi;
using GHIElectronics.TinyCLR.Devices.Gpio;
using GHIElectronics.TinyCLR.Pins;

ST7735Controller st7735 = null;
void InitDisplay() {
    // Display Get Ready ////////////////////////////////////
    var spi = SpiController.FromName(FEZBit.SpiBus.Display);
    var gpio = GpioController.GetDefault();

    st7735 = new ST7735Controller(
        spi.GetDevice(ST7735Controller.GetConnectionSettings
        (SpiChipSelectType.Gpio,
        gpio.OpenPin(FEZBit.GpioPin.DisplayChipselect))),
        gpio.OpenPin(FEZBit.GpioPin.DisplayRs),
        gpio.OpenPin(FEZBit.GpioPin.DisplayReset)
    );

    var backlight = gpio.OpenPin(FEZBit.GpioPin.Backlight);
    backlight.SetDriveMode(GpioPinDriveMode.Output);
    backlight.Write(GpioPinValue.High);
    st7735.SetDataAccessControl(true, true, false, false); //Rotate the screen.
    st7735.SetDrawWindow(0, 0, 160, 128);
    st7735.Enable();
}

InitDisplay();
```

The driver needs to know what SPI bus to use as well as where the other control pins are connected. There is also a backlight pin that you can use to control the backlight. By the way, this pin is PWM capable if you want to dim the backlight.

TinyCLR includes graphics support, which is detailed in the graphics section in this guide. However, here is example code to put something on the screen. Just note that this code needs to be added to the above display initialization code.

This example uses BasicGraphics found in the GHIElectronics.TinyCLR.Drivers.BasicGraphics NuGet/namespace.

```
var basicGfx = new BasicGraphics(160, 128, ColorFormat.Rgb565);
var colorBlue = BasicGraphics.ColorFromRgb(0,0,255);
var colorGreen = BasicGraphics.ColorFromRgb(0, 255,0);
var colorRed = BasicGraphics.ColorFromRgb(255, 0,0);
var colorWhite = BasicGraphics.ColorFromRgb(255, 255, 255);

basicGfx.Clear();

basicGfx.DrawString("TinyCLR OS!", colorGreen, 15, 15, 2, 1);
basicGfx.DrawString("FEZ Bit", colorBlue, 35, 40, 2, 2);
basicGfx.DrawString("SC20100", colorRed, 35, 60, 2, 2);

Random color = new Random();
for (var i = 20; i < 140; i++)
    basicGfx.DrawCircle((uint)color.Next(), i, 100, 15);

st7735.DrawBuffer(basicGfx.Buffer);
```

I2C

I2C was developed by Phillips to allow multiple chipsets to communicate on a 2-wire bus in home consumer devices, mainly TV sets. Like SPI, I2C have a master and one or more slaves on the same data bus. However, instead of selecting the slaves using a digital pin like SPI, I2C uses software addressing. Before data is transferred, the master sends out a 7-bit address of the slave device it wants to communicate with. It also sends a bit indicating if the master wants to send or receive data. The slave that sees its address on the bus will acknowledge its presence. At this point, the master can send/receive data. The master will start data transfers with “start condition” before it sends any address or data, and then end it with “stop” condition.

There is also another condition in I2C that is called restart condition. This is when two transactions need to be glued together as one. For example, a device may respond to a command. The command needs to be sent in a write transaction and the response comes back in a read transaction. Using a write transaction separate from read will not work. And for this, there is a WriteRead method.

Finally, while any clock rate can be used, the specifications call for 100KHz for standard mode and 400KHz for fast mode.

FEZ Bit includes an I2C accelerometer at slave address 0x1D. This accelerometer responds to 0x0F (device ID request) with 0x41, which is the device ID. Having an ID response is a common thing in I2C to verify there is a slave and it is the right slave.

The I2C support is found under GHIElectronics.TinyCLR.Devices.I2c NuGet/namespace.

```
using System.Diagnostics;
using GHIElectronics.TinyCLR.Devices.I2c;
using GHIElectronics.TinyCLR.Pins;

var settings = new I2cConnectionSettings(0x1D, 100_000);
var controller = I2cController.FromName(FEZBit.I2cBus.Accelerometer);
var device = controller.GetDevice(settings);

var readByte = new byte[1];
device.WriteRead(new byte[] { 0x0F }, readByte);
if (readByte[0] == 0x41)
    Debug.WriteLine("Accel Detected!");
else
    Debug.WriteLine("Something is wrong!");
```

The driver for the accelerometer can be a good full reference for I2C use. The accelerometer is LIS2HH12 from ST Micro. Drivers are found under GHIElectronics.TinyCLR.Drivers.STMicroelectronics.LIS2HH12 NuGet/namespace.

```
using System.Threading;
using System.Diagnostics;
using GHIElectronics.TinyCLR.Devices.I2c;
using GHIElectronics.TinyCLR.Pins;
using GHIElectronics.TinyCLR.Drivers.STMicroelectronics.LIS2HH12;

var controller = I2cController.FromName(FEZBit.I2cBus.Accelerometer);
var lis2hh12 = new LIS2HH12Controller(controller);

while (true) {
    Debug.WriteLine("x > " + lis2hh12.X);
    Thread.Sleep(50);
}
```

BrainBot also uses I2C. There is a chip on the robot that listens to I2C commands at slave address 0x01 to control the motors and the headlights.

The commands are very simple. Send 0x01 followed by Red, Green and Blue to control the headlights.

```
using System.Threading;
using GHIElectronics.TinyCLR.Devices.I2c;
using GHIElectronics.TinyCLR.Pins;

var settings = new I2cConnectionSettings(0x01, 100_000);
var controller = I2cController.FromName(FEZBit.I2cBus.Edge);
var device = controller.GetDevice(settings);
void SetHeadlight(byte red, byte green, byte blue) {
    var b4 = new byte[4];
    b4[0] = 0x01;
    b4[1] = red;
    b4[2] = green;
    b4[3] = blue;
    device.Write(b4);
}

while (true) {
    SetHeadlight(200, 0, 0);
    Thread.Sleep(300);
    SetHeadlight(0, 0, 200);
    Thread.Sleep(300);
}
```

The motors are controlled by sending 0x02 followed by left, left-invert, right, right-invert.

```
using System.Threading;
using GHIElectronics.TinyCLR.Devices.I2c;
using GHIElectronics.TinyCLR.Pins;

var settings = new I2cConnectionSettings(0x01, 100_000);
var controller = I2cController.FromName(FEZBit.I2cBus.Edge);
var device = controller.GetDevice(settings);

void SetMotorSpeed(double left, double right) {
    var b5 = new byte[5];
    left = left / 100;right = right / 100;
    b5[0] = 0x02;
    if (left > 0) {
        b5[1] = (byte)(left * 255);
        b5[2] = 0x00;
    }
    else {
        left *= -1;
        b5[1] = 0x00;
        b5[2] = (byte)(left * 255);
    }
    if (right > 0) {
        b5[3] = (byte)(right * 255);
        b5[4] = 0x00;
    }
    else {
        right *= -1;
        b5[3] = 0x00;
        b5[4] = (byte)(right * 255);
    }
    device.Write(b5);
}

while (true) {
    SetMotorSpeed(90,-10);
    Thread.Sleep(2000);
    SetMotorSpeed(30, 90);
    Thread.Sleep(2000);
}
```

CAN

Controller Area Network is a very common interface in industrial control and automotive. CAN is very robust and works very well in noisy environments at high speeds. All error checking and recovery methods are done automatically on the hardware. CAN sends messages that consist of an ID followed by 8 data bytes. Messages are more complex than that, but we are keeping it simple for this introduction.

A transceiver needs to be added to the CAN pins. There are multiple types of transceivers, and they are typically not compatible. CAN is available on FEZ Bit, P12: CAN-RX and P11: CAN-TX. Accessing those pins would require a breakout board plus a transceiver needs to be added.

The bit timing on CAN is sophisticated with several configurations. There are several tools to help in calculating the right settings.

We will end this section with a simple example that sends a CAN message. More info can be found at https://docs.ghielectronics.com/software/tinyclr/tutorials/can.html and https://en.wikipedia.org/wiki/CAN_bus

The CAN drivers are found under GHIElectronics.TinyCLR.Devices.Can NuGet/namespace.

```
using GHIElectronics.TinyCLR.Devices.Can;
using GHIElectronics.TinyCLR.Pins;

var can = CanController.FromName(SC20100.CanBus.Can1);
var propagationPhase1 = 13;
var phase2 = 2;
var baudratePrescaler = 3;
var synchronizationJumpWidth = 1;
var useMultiBitSampling = false;
can.SetNominalBitTiming(new CanBitTiming(propagationPhase1, phase2, baudratePrescaler,
    synchronizationJumpWidth, useMultiBitSampling));

var message = new CanMessage() {
    Data = new byte[] { 0x48, 0x65, 0x6C, 0x6C, 0x6F, 0x2E, 0x20, 0x20 },
    ArbitrationId = 0x11,
    Length = 6,
    RemoteTransmissionRequest = false,
    ExtendedId = false,
    FdCan = false,
    BitRateSwitch = false
};

can.WriteMessage(message);
```

DIGITAL SIGNALS

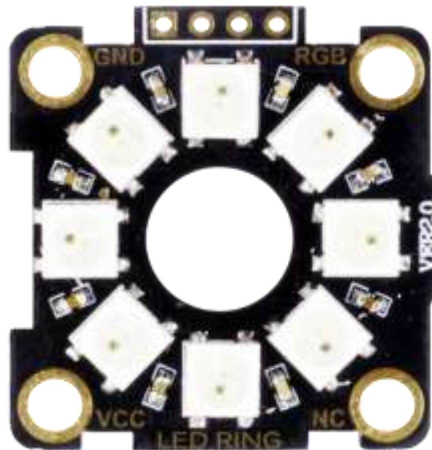
While everything on a digital system is a digital signal, we are here referring to controlling devices with specialized signals that are not one of the standard busses.

ADDRESSABLE LEDS

The market is flooded with addressable LEDs. These are LEDs that have very tiny microcontrollers built into each single LED. This allows a system, like FEZ Panda, to control hundreds of LEDs using a single pin.

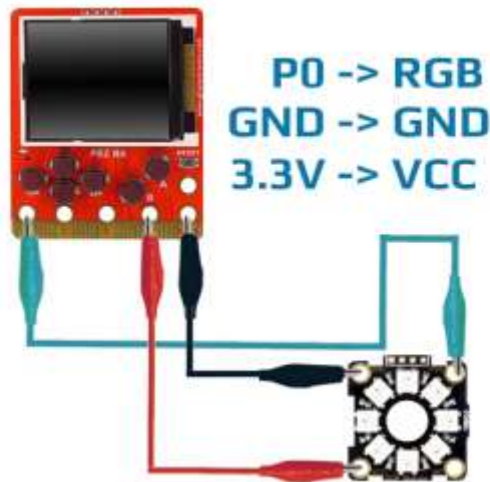
This close-up picture shows the micro, which is connected individually to three RGB LEDs (Red, Green and Blue).

TinyCLR support LP8006, APA102C and the most common WS2812. You already have these LEDs in the kit on the LED Ring module.



Even though this module has 8 LEDs and each LED has 3 LED elements inside, you can control all these 24 LEDs using a single pin! The digital signal going out on the pin is very specific and requires great accuracy.

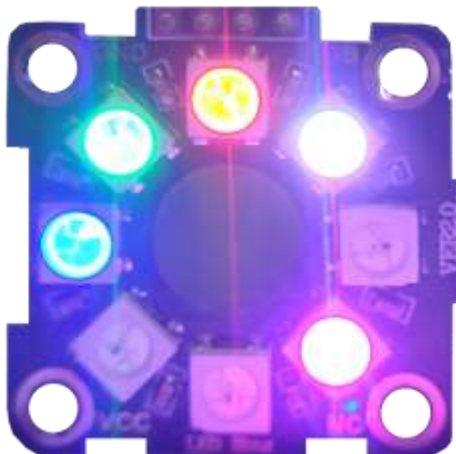
Connect the FEZ Bit to the LED Ring Module as shown in the diagram below.



The drivers are found in the GHIElectronics.TinyCLR.Drivers.Worldsemi.WS2812 NuGet/namespace.

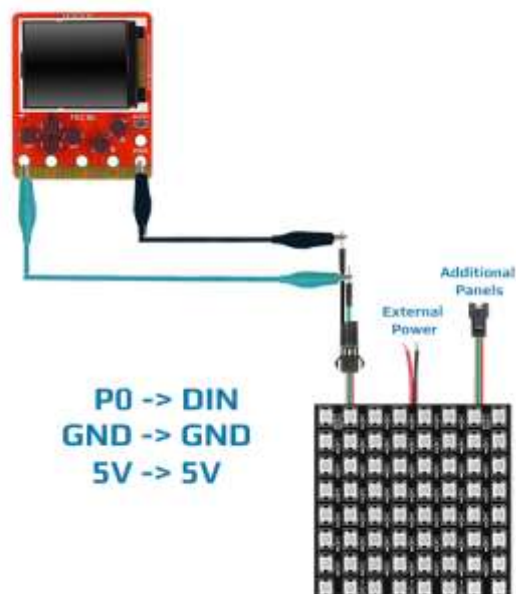
```
using GHIElectronics.TinyCLR.Drivers.Worldsemi.WS2812;
using GHIElectronics.TinyCLR.Devices.Gpio;
using GHIElectronics.TinyCLR.Pins;

const int NUM_LED = 8;
var pin = GpioController.GetDefault().OpenPin(FEZBit.GpioPin.P0);
var leds = new WS2812Controller(pin, NUM_LED, WS2812Controller.DataFormat.rgb888);
leds.SetColor(0, 0xFF, 0, 0); // red
leds.SetColor(1, 0, 0xFF, 0); // green
leds.SetColor(2, 0, 0, 0xFF); // blue
leds.SetColor(5, 0xff, 0, 0xFF); // purple
leds.SetColor(7, 0xFF, 0xFF, 0xFF); // white
leds.Flush();
```

To prevent the LEDs from flickering, the driver buffers up the desired values and only sends them to the LEDs when Flush method is called.



The driver supports RGB888 to give the max possible color option, but it also supports RGB565 to allow TinyCLR graphics to be placed on LED matrices.

Those matrices are common and can be wired easily to FEZ Bit. TinyCLR graphics libraries can me mapped to produce the graphics on these matrices.



The same LEDs are used on the BrainBot, the 2 rear LEDs. And because these are smart LEDs, they can be controlled individually. The LEDs on BrainBot are connected to P12.

```
using GHIElectronics.TinyCLR.Drivers.Worldsemi.WS2812;

const int NUM_LED = 2;
var pin = GpioController.GetDefault().OpenPin(FEZBit.GpioPin.P12);
var leds = new WS2812Controller(pin, NUM_LED, WS2812Controller.DataFormat.rgb888);
leds.SetColor(0, 0xFF, 0, 0); // red
leds.SetColor(1, 0, 0, 0xFF); // blue
leds.Flush();
```

Let's make a police car!

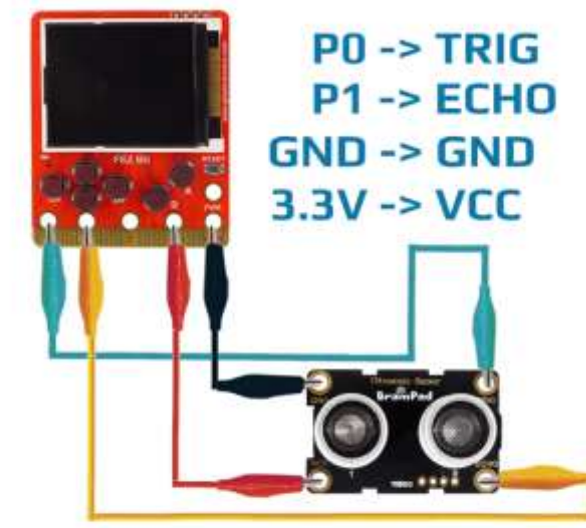
```
using System.Threading;
using GHIElectronics.TinyCLR.Drivers.Worldsemi.WS2812;

const int NUM_LED = 2;
var pin = GpioController.GetDefault().OpenPin(FEZBit.GpioPin.P12);
var leds = new WS2812Controller(pin, NUM_LED, WS2812Controller.DataFormat.rgb888);

while(true){
    leds.SetColor(0, 0xFF, 0, 0); // red
    leds.SetColor(1, 0, 0, 0xFF); // blue
    leds.Flush();
    Thread.Sleep(300);
    leds.SetColor(0, 0, 0, 0xFF); // blue
    leds.SetColor(1, 0xFF, 0, 0); // red
    leds.Flush();
    Thread.Sleep(300);

}
```

ULTRASONIC SENSOR

There are many ways to measure, but the most common sensor uses ultrasound to measure distance by sending a pulse and measuring the time needed for the response to come back. The pulse is of a higher frequency than we can hear and so the sensor seems quiet, but it is not!

Sending a pulse starts with TRIG pin, the trigger. The response with the distance comes back on the ECHO pin.

Connect the FEZ Bit to the Ultrasonic Sensor Module as shown in the diagram below.



To handle the tight timing, there is a nice library called *signals* that handles multiple tasks, including PulseFeedback. The driver is found GHIElectronics.TinyCLR.Devices.Signals NuGet/namespace.

```
using System;
using System.Threading;
using GHIElectronics.TinyCLR.Devices.Signals;
using GHIElectronics.TinyCLR.Pins;
using GHIElectronics.TinyCLR.Devices.Gpio;
using System.Diagnostics;

var trig = GpioController.GetDefault().OpenPin(FEZBit.GpioPin.P0);
var echo = GpioController.GetDefault().OpenPin(FEZBit.GpioPin.P1);
var pulseFeedback = new PulseFeedback(trig, echo, PulseFeedbackMode.EchoDuration) {
    DisableInterrupts = false,
    Timeout = TimeSpan.FromSeconds(1),
    PulseLength = TimeSpan.FromTicks(100),
    PulseValue = GpioPinValue.High,
    EchoValue = GpioPinValue.High,
};
```

The robot also has a distance sensor that connects TRIG to P16 and ECHO to P15.

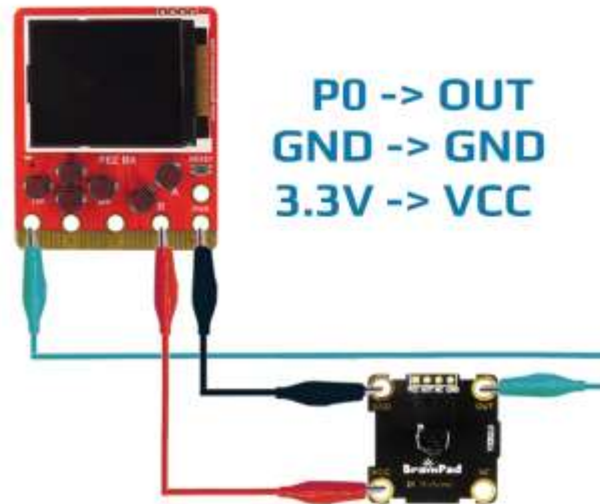
```
using System;
using System.Threading;
using GHIElectronics.TinyCLR.Devices.Signals;
using GHIElectronics.TinyCLR.Pins;
using GHIElectronics.TinyCLR.Devices.Gpio;
using System.Diagnostics;

var trig = GpioController.GetDefault().OpenPin(FEZBit.GpioPin.P16);
var echo = GpioController.GetDefault().OpenPin(FEZBit.GpioPin.P15);
var pulseFeedback = new PulseFeedback(trig, echo, PulseFeedbackMode.EchoDuration) {
    DisableInterrupts = false,
    Timeout = TimeSpan.FromSeconds(1),
    PulseLength = TimeSpan.FromTicks(100),
    PulseValue = GpioPinValue.High,
    EchoValue = GpioPinValue.High,
};
```

IR REMOTE CONTROL

Infrared (IR) Remote controls are old technology that is still in use today due to its simplicity. The remote sends a pulse of IR light that we do not see, then an IR receiver detects those pulses and uses them to decode a meaningful signal. TinyCLR includes an NCR standard signals decoder. This works with the remote found with BrainClip and the BrainBot.

To read/see the IR signal, we need the IR Receiver module.

Connect the FEZ Bit to the IR Receiver module as shown in the diagram below.



The drivers are found in GHIElectronics.TinyCLR.Drivers.Infrared NuGet/namespace.

```
using System.Threading;
using System.Diagnostics;
using GHIElectronics.TinyCLR.Pins;
using GHIElectronics.TinyCLR.Drivers.Infrared;
using GHIElectronics.TinyCLR.Devices.Gpio;

var recievePin = GpioController.GetDefault().OpenPin(FEZBit.GpioPin.P0);
var ir = new NecIRDecoder(recievePin);

ir.OnDataReceivedEvent += Ir_OnDataRecievedEvent;
ir.OnRepeatEvent += Ir_OnRepeatEvent;
Debug.WriteLine("System is ready!");
Thread.Sleep(Timeout.Infinite);

void Ir_OnDataRecievedEvent(byte address, byte command) {
    Debug.WriteLine("A: " + address + " C: " + command);
}
void Ir_OnRepeatEvent() {
    Debug.WriteLine("Repeat!");
}
```

Running the code will show an output when there is a remote button press.

Note that the remote has a little plastic insert to keep the battery disconnected. Pull the plastic insert to use the remote.

```
System is ready!
A: 0 C: 9
Repeat!
Repeat!
A: 0 C: 0
Repeat!
A: 0 C: 13
Repeat!
A: 0 C: 9
```

On the BrainBot the IR Receiver is located right under the ultrasonic sensor, at the very front.

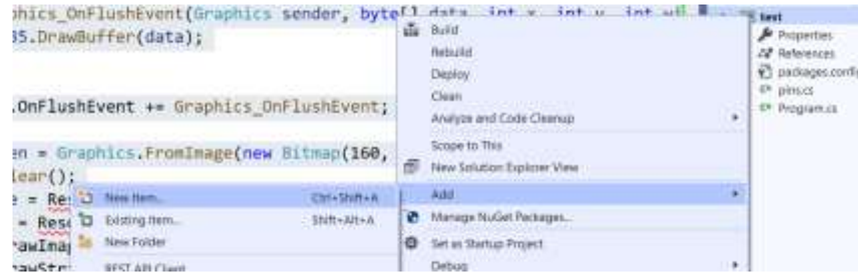
The sensor is connected to pin P8.

```
using System.Threading;
using System.Diagnostics;
using GHIElectronics.TinyCLR.Pins;
using GHIElectronics.TinyCLR.Drivers.Infrared;
using GHIElectronics.TinyCLR.Devices.Gpio;

var recievePin = GpioController.GetDefault().OpenPin(FEZBit.GpioPin.P8);
var ir = new NecIRDecoder(recievePin);

ir.OnDataReceivedEvent += Ir_OnDataRecievedEvent;
ir.OnRepeatEvent += Ir_OnRepeatEvent;
Debug.WriteLine("System is ready!");
Thread.Sleep(Timeout.Infinite);

void Ir_OnDataRecievedEvent(byte address, byte command) {
    Debug.WriteLine("A: " + address + " C: " + command);
}
void Ir_OnRepeatEvent() {
    Debug.WriteLine("Repeat!");
}
```

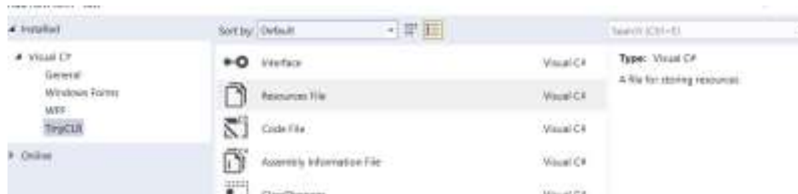
LOADING RESOURCES

A resource is a data that is included with the application. If an application depends on a file (image, icon, font, sound), then it is a good idea to add this as a resource. This is especially important when using graphics.

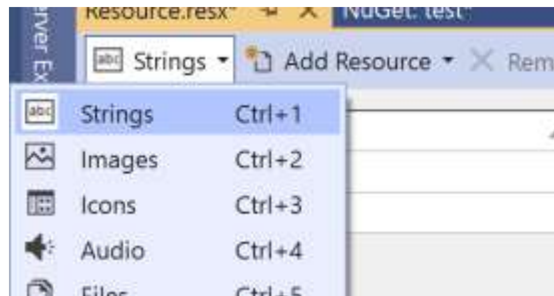
To add a resource file, right click on the project and select **Add -> New Item…** from the menu.



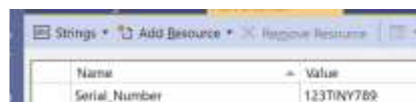
We now can select TinyCLR from under Visual C# and then select **Resource File**. Name the file Resources.resx.



Click on the Resources.resx file inside the Solution Explorer of your project. We will start by adding a string resource since this is the easiest one. Select Strings from the drop-down tab at the top left corner. This will show us the all the ‘String’ resources in our project. By default, ‘String1’ is already in the table. We can modify this one. If we want to add more, we can click on the ‘Add Resource’ drop-down.



Click on the ‘String1’ and let’s rename to what we want. Let’s name it *Serial_Number* and give it a value. Save the file.



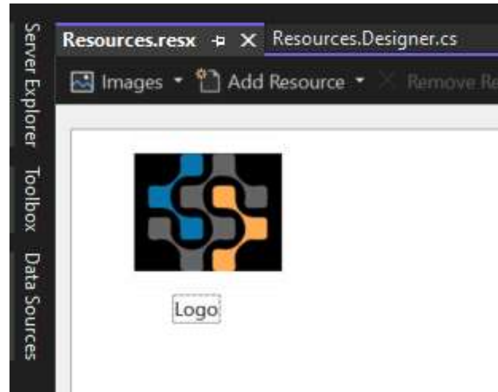
Once saved, we can access that string resource in the code.

```
using System.Diagnostics;

Debug.WriteLine(Resources.GetString(
    Resources.StringResources.Serial_Number));
```

Note how the resource file has a namespace that is by default same as the project’s name. If current file and resource do not have the same name space, or if you are using top-level-statements, then the namespace should be added. The namespace in my project is *test*, for example.

```
using System.Diagnostics;

Debug.WriteLine(test.Resources.GetString(
    test.Resources.StringResources.Serial_Number));
```

To add an image, we can drag-and-drop it into resources.



Fonts and other resources can be added as well. The display chapter will show more details.

DISPLAYS

TinyCLR has a strong display and graphics support allowing for multiple options. FEZ Bit has a nice 160x128 color display and, therefore, this book will focus on that. There are 2 ways to use this display. One option is by using the native support, with fonts and image and UI support. The other option is by using BasicGraphics, which is a simple lean C# library. In either case, the first thing we need is to bring in the display drivers and initialize the display. This was covered in the SPI chapter, but here is the display initialization code again.

The display driver is found in the GHIElectronics.TinyCLR.Drivers.Sitronix.ST7735 NuGet/namespace. This will automatically pull in the needed SPI and GPIO.

```
using System;
using GHIElectronics.TinyCLR.Devices.Spi;
using GHIElectronics.TinyCLR.Devices.Gpio;
using GHIElectronics.TinyCLR.Drivers.Sitronix.ST7735;
using GHIElectronics.TinyCLR.Pins;

ST7735Controller st7735 = null;
void InitDisplay() {
    // Display Get Ready ////////////////////////////////////
    var spi = SpiController.FromName(FEZBit.SpiBus.Display);
    var gpio = GpioController.GetDefault();

    st7735 = new ST7735Controller(
        spi.GetDevice(ST7735Controller.GetConnectionSettings
        (SpiChipSelectType.Gpio,
        gpio.OpenPin(FEZBit.GpioPin.DisplayChipselect))), //CS pin.
        gpio.OpenPin(FEZBit.GpioPin.DisplayRs), //RS pin.
        gpio.OpenPin(FEZBit.GpioPin.DisplayReset) //RESET pin.
    );

    var backlight = gpio.OpenPin(FEZBit.GpioPin.Backlight);
    backlight.SetDriveMode(GpioPinDriveMode.Output);
    backlight.Write(GpioPinValue.High);

    st7735.SetDataAccessControl(true, true, false, false); //Rotate the screen.
    st7735.SetDrawWindow(0, 0, 160, 128);
    st7735.Enable();
}
```

Once we have an initialized display, we can put graphics on it.

BASIC GRAPHICS

The BasicGraphics library is a simple library that is written completely in C#. It works well for driving small displays and LED matrices. It includes two 5x8 and 5x5 fonts and supports drawing basic elements, like lines and circles.

There are two ways to use BasicGraphcis. The first way is by inheriting the BasicGraphics class and overriding the SetPixel method. This method requires zero memory and gives the user full control. The second way is by allowing BasicGraphics to create an internal buffer and do all the drawing onto. This is a preferred method when using FEZ Bit. We let BasicGraphics handle all the drawing and buffering in its own buffer. We then transfer the drawing buffer to the display. Here is a quick example. Do not forget to have the Display initialization code first. The BasicGraphics library is found in GHIElectronics.TinyCLR.Drivers.BasicGraphics NuGet/namespace.

```
var basicGfx = new BasicGraphics(160, 128, ColorFormat.Rgb565);
var colorBlue = BasicGraphics.ColorFromRgb(0,0,255);
basicGfx.DrawString("FEZ Bit", colorBlue, 35, 40, 2, 2);

st7735.DrawBuffer(basicGfx.Buffer);
```

The first three lines are creating the BasicGraphics object with the display's size of 160x128, and RGB565 color format. We then create a color and draw some text with that color.

The last line is sending the BasicGraphics buffer to the display.

For better demonstration, let us read the sound sensor module and show the sound level on the screen as a graph!

The connections are like usual. GND to GND, VCC to 3V and SIG is the signal, which we will connect to P0 pad.

The graph will be done by drawing lines that start from the sound level scaled to 105 on top, all the way to pixel 105 on the bottom. We will then shift the line one pixel to the right. Once the line reaches the very right of the screen at 160, we would clear the screen and start all over.

```
InitDisplay();

var basicGfx = new BasicGraphics(160, 128, ColorFormat.Rgb565);
var colorGreen = BasicGraphics.ColorFromRgb(0, 255, 0);
var colorRed = BasicGraphics.ColorFromRgb(255, 0, 0);
var controller = AdcController.FromName(FEZBit.Adc.Controller3.Id);
var p0 = controller.OpenChannel(FEZBit.Adc.Controller3.P0);

var x = 200;
while (true) {
    x++;
    if (x > 160) {
        basicGfx.Clear();
        x = 0;
        basicGfx.DrawString("Sound Level", colorRed, 18, 110, 2, 2);
    }
    var y = p0.ReadRatio() * 105;
    basicGfx.DrawLine(colorGreen, x, (int)y, x, 105);
    st7735.DrawBuffer(basicGfx.Buffer);
    Thread.Sleep(10);
}
```

NATIVE GRAPHICS

The native support is a lot more powerful than BasicGraphics. It does not have built in fonts, but you can create any custom fonts that get converted from TrueType to TinyFont. It also supports BMP, JPG and GIF images and includes a UI library. This is the library of choice with lager displays, especially when user interface and touch screen is necessary. FEZ Bit work well with native and basic graphics.

To use native graphics we will need GHIElectronics.TinyCLR.Devices.Display and GHIElectronics.TinyCLR.Drawing. The namespace needed is System.Drawing. You'll also have to add the InitDisplay() function shown earlier to initialize the display.

We first need to subscribe to a flush event, where then we can send the data to be flushed to the display. The rest is the same on any other display type.

```
using System.Drawing;

void Graphics_OnFlushEvent(Graphics sender, byte[] data, int x, int y, int width, int
height, int originalWidth) {
    st7735.DrawBuffer(data);
}

InitDisplay();

Graphics.OnFlushEvent += Graphics_OnFlushEvent;

var screen = Graphics.FromImage(new Bitmap(160, 128));
screen.Clear();
screen.FillEllipse(new SolidBrush(System.Drawing.Color.FromArgb
    (255, 255, 0, 0)), 0, 0, 80, 64);
screen.FillEllipse(new SolidBrush(System.Drawing.Color.FromArgb
    (255, 0, 0, 255)), 80, 0, 80, 64);
screen.FillEllipse(new SolidBrush(System.Drawing.Color.FromArgb
    (128, 0, 255, 0)), 40, 0, 80, 64);

screen.DrawRectangle(new Pen(Color.Yellow), 10, 80, 40, 25);
screen.DrawEllipse(new Pen(Color.Purple), 60, 80, 40, 25);
screen.FillRectangle(new SolidBrush(Color.Teal), 110, 80, 40, 25);

screen.DrawLine(new Pen(Color.White), 10, 127, 150, 127);
screen.SetPixel(80, 92, Color.White);

screen.Flush();
```

IMAGES

Native graphics supports JPG, GIF, and BMP image types. The image can be loaded from SD card, or from a resource. Here is an example to get you started, but see the resources section to learn about adding resources to your program.

```
void Graphics_OnFlushEvent(Graphics sender, byte[] data, int x, int y, int width, int
height, int originalWidth) {
    st7735.DrawBuffer(data);
}

InitDisplay();

Graphics.OnFlushEvent += Graphics_OnFlushEvent;

var screen = Graphics.FromImage(new Bitmap(160, 128));
screen.Clear();
var image = Resources.GetBitmap(Resources.BitmapResources.Image);

screen.DrawImage(image, 0, 0);

screen.Flush();
```

FONTS

Standard fonts con be converted from TrueType to TinyFonts in order to be used with native graphics. More details on the converter tool are available on the docs https://docs.ghielectronics.com/software/tinyclr/tutorials/font-support.html .

ARTIFICIAL INTELLIGENCE

In this chapter, we will do a simple line follower that stops when it detects objects.

Add a new TinyCLR Class file to your project with the code below. We only need Signals, I2C, WS2812, and GPIO NuGets.

```
using System;
using GHIElectronics.TinyCLR.Devices.I2c;
using GHIElectronics.TinyCLR.Devices.Gpio;
using GHIElectronics.TinyCLR.Devices.Signals;
using GHIElectronics.TinyCLR.Drivers.Worldsemi.WS2812;

namespace BrainPad.BrainBot {
    class BrainBotController {
        private I2cDevice i2c;
        private GpioPin leftLineSensor, rightLineSensor;
        private byte[] b4 = new byte[4];
        private byte[] b5 = new byte[5];
        private PulseFeedback pulseFeedback;
        private WS2812Controller tailLight;
        public BrainBotController(I2cController i2cController, GpioPin leftLineSensor, GpioPin      rightLineSensor, GpioPin
distanceTrigger,
       GpioPin distanceEcho, GpioPin tailLight){
            this.i2c = i2cController.GetDevice(new I2cConnectionSettings(0x01, 100_000));
            this.leftLineSensor = leftLineSensor;
            this.leftLineSensor.SetDriveMode(GpioPinDriveMode.Input);
            this.rightLineSensor = rightLineSensor;
            this.rightLineSensor.SetDriveMode(GpioPinDriveMode.Input);
            this.tailLight = new WS2812Controller(tailLight, 2, WS2812Controller.DataFormat.rgb888);
            this.pulseFeedback = new PulseFeedback(distanceTrigger, distanceEcho, PulseFeedbackMode.EchoDuration) {
                DisableInterrupts = false,
                Timeout = TimeSpan.FromSeconds(1),
                PulseLength = TimeSpan.FromTicks(100),
                PulseValue = GpioPinValue.High,
                EchoValue = GpioPinValue.High,
            };
        }
        public void SetMotorSpeed(double left, double right) {
            this.b5[0] = 0x02;
            left = left / 100;
            right = right / 100;
            if (left > 0) {
                this.b5[1] = (byte)(left * 255);
                this.b5[2] = 0x00;
            }
            else {
                left *= -1;
                this.b5[1] = 0x00;
                this.b5[2] = (byte)(left * 255);
            }
            if (right > 0) {
                this.b5[3] = (byte)(right * 255);
                this.b5[4] = 0x00;
            }
            Else {
                right *= -1;
                this.b5[3] = 0x00;
                this.b5[4] = (byte)(right * 255);
            }
            this.i2c.Write(this.b5);
        }
        public void SetHeadlight(int red, int green, int blue) {
            this.b4[0] = 0x01;
            this.b4[1] = (byte)red;
            this.b4[2] = (byte)green;
            this.b4[3] = (byte)blue;
            this.i2c.Write(this.b4);
        }
        public void SetTaillight(bool isRight, int red, int blue, int green) {
            tailLight.SetColor(isRight ? 0 : 1, (byte)red, (byte)green, (byte)blue);
            tailLight.Flush();
        }
        public int ReadDistance(){
            var time = this.pulseFeedback.Trigger();
            var microsecond = time.TotalMilliseconds * 1000.0;
            var distance = microsecond * 0.036 / 2;
            return (int)distance;
        }
        public int ReadLineSensor() {
            int r = 0;
            if (this.rightLineSensor.Read() == GpioPinValue.Low)
                r |= 0x01;
            if (this.leftLineSensor.Read() == GpioPinValue.Low)
                r |= 0x02;
            return r;
        }
    }
}
```

If you get an exception when calling i2c.Write(), the FEZ Bit may not be properly seated in the robot.

We can now use the robot a bit easier. The main program file can define the pins and control the robot.

```
using System.Threading;
using BrainPad.BrainBot;
using GHIElectronics.TinyCLR.Devices.Gpio;
using GHIElectronics.TinyCLR.Devices.I2c;
using GHIElectronics.TinyCLR.Pins;

var gpio = GpioController.GetDefault();
var bot = new BrainBotController(
    I2cController.FromName(FEZBit.I2cBus.Edge),
    gpio.OpenPin(FEZBit.GpioPin.P13),
    gpio.OpenPin(FEZBit.GpioPin.P14),
    gpio.OpenPin(FEZBit.GpioPin.P16),
    gpio.OpenPin(FEZBit.GpioPin.P15),
    gpio.OpenPin(FEZBit.GpioPin.P12)
    );

new Thread(() => {
    while (true) {
        bot.SetTailLight(false, 100, 0, 0);
        bot.SetTailLight(true, 0, 100, 0);
        bot.SetHeadlight(100, 0, 0);
        Thread.Sleep(200);
        bot.SetTailLight(false, 0, 100, 0);
        bot.SetTailLight(true, 100, 0, 0);
        bot.SetHeadlight(0, 0, 100);
        Thread.Sleep(200);
    }
}).Start();

// add your code here...
Thread.Sleep(Timeout.Infinite);
```

Let's move the robot. Modify the above code by adding the below code. Do the same for all the following examples in this chapter.

```
// add your code here...

while (true) {
    while (true) {
        bot.SetMotorSpeed(-90, 90);
        Thread.Sleep(1000);
        bot.SetMotorSpeed(50, 50);
        Thread.Sleep(500);
    }
}
```

By this point, you maybe be concerned because the robot keeps trying to run away! Can we add code to the top of what we just added to stop the robot on power up and only run after a button is pressed?

```
bot.SetMotorSpeed(0, 0);
var btnA = gpio.OpenPin(FEZBit.GpioPin.ButtonA);
btnA.SetDriveMode(GpioPinDriveMode.InputPullUp);
while (btnA.Read() == GpioPinValue.High)
    Thread.Sleep(10);
// add your code here...
```

Now you can stop the robot anytime by pressing the reset button on FEZ Bit, and then let it run by pressing the A button.

The next step is to drive the robot to follow a line. We are using a very simple fixed speed to move the robot and stop one of the wheels when a line is detected on one of the sensors to make the robot turn back onto the line.

```
// add your code here...

while (true) {
    while (true) {
        var value = bot.ReadLineSensor();
        if (value == 3 || value == 0) {
            bot.SetMotorSpeed(25, 25);
        }
        if (value == 2) {
            bot.SetMotorSpeed(25, 0);
        }
        if (value == 1) {
            bot.SetMotorSpeed(0, 0.25);
        }
    }
}
```

The robot should now be going around the line. We can make the robot a little bit smarter by checking the distance sensor for obstacles.

```
// add your code here...

while (true) {
    while (true) {
        while(bot.ReadDistance() < 10) {
            bot.SetMotorSpeed(0, 0);
        }
        var value = bot.ReadLineSensor();
        if (value == 3 || value == 0) {
            bot.SetMotorSpeed(25, 25);
        }
        if (value == 2) {
            bot.SetMotorSpeed(25, 0);
        }
        if (value == 1) {
            bot.SetMotorSpeed(0, 25);
        }
    }
}
```

NETWORKING

Networks are an essential part of our lives both at home and work. These networks use standard ways of transferring data. You might have heard of TCP/IP, DNS, DHCP, IP, ICMP, TCP, UDP, PPP...and many more!

TinyCLR has full networking support with ".NET sockets" and includes proper TLS security. Networking is possible over WiFi, Ethernet, and PPP (for mobile modems). We will focus on WiFi since FEZ Bit includes built-in WiFi module.

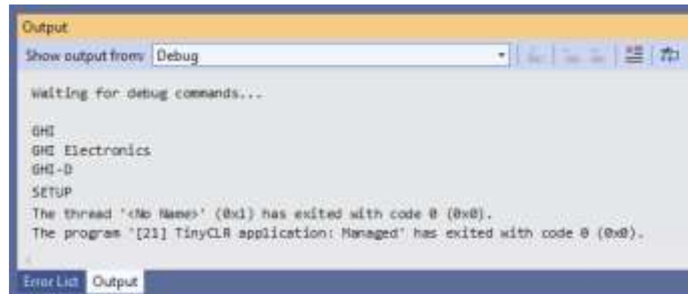
WIFI SETUP

The first step is to let the system know what pins are used for the WiFi module. We will need to add the GHIElectronics.TinyCLR.Devices.Network NuGet/namespace. This library handles the networking interfaces, such as WiFi and Ethernet. When added, the library will internally and automatically pull in GHIElectronics.TinyCLR.Networking, which handles the networking protocols. Anytime we add Wifi to a program, we'll need to call the InitWifiModule() function.

```
using System;
using System.Diagnostics;
using System.Threading;
using GHIElectronics.TinyCLR.Devices.Gpio;
using GHIElectronics.TinyCLR.Devices.Network;
using GHIElectronics.TinyCLR.Devices.Spi;
using GHIElectronics.TinyCLR.Pins;

var networkController = NetworkController.FromName
    (SC20100.NetworkController.ATWinc15x0);

void InitWifiModule() {

//Setup Pins
var enablePinNumber = FEZBit.GpioPin.WiFiEnable;
var chipSelectPinNumber = FEZBit.GpioPin.WiFiChipselect;
var irqPinNumber = FEZBit.GpioPin.WiFiInterrupt;
var resetPinNumber = FEZBit.GpioPin.WiFiReset;
var spiControllerName = FEZBit.SpiBus.WiFi;

var gpio = GpioController.GetDefault();
var enablePin = gpio.OpenPin(enablePinNumber);
enablePin.SetDriveMode(GpioPinDriveMode.Output);
enablePin.Write(GpioPinValue.High);

SpiNetworkCommunicationInterfaceSettings netInterfaceSettings =
    new SpiNetworkCommunicationInterfaceSettings();

var settings = new SpiConnectionSettings() {
    ChipSelectLine = gpio.OpenPin(chipSelectPinNumber),
    ClockFrequency = 4000000,
    Mode = SpiMode.Mode0,
    ChipSelectType = SpiChipSelectType.Gpio,
    ChipSelectHoldTime = TimeSpan.FromTicks(10),
    ChipSelectSetupTime = TimeSpan.FromTicks(10)
};

netInterfaceSettings.SpiApiName = spiControllerName;
netInterfaceSettings.GpioApiName = SC20260.GpioPin.Id;
netInterfaceSettings.SpiSettings = settings;
netInterfaceSettings.InterruptPin = gpio.OpenPin(irqPinNumber);
netInterfaceSettings.InterruptEdge = GpioPinEdge.FallingEdge;
netInterfaceSettings.InterruptDriveMode = GpioPinDriveMode.InputPullUp;
netInterfaceSettings.ResetPin = gpio.OpenPin(resetPinNumber);
netInterfaceSettings.ResetActiveState = GpioPinValue.Low;

networkController.SetCommunicationInterfaceSettings(netInterfaceSettings);
}
```

We can now scan for the available WiFi networks. This optional service and many others are not "networking" related but are WiFi module specific. They are available from the GHIElectronics.TinyCLR.Drivers.Microchip.Winc15x0 NuGet/namespace.

```
using GHIElectronics.TinyCLR.Drivers.Microchip.Winc15x0;

string[] ssidList = Winc15x0Interface.Scan();
foreach (var s in ssidList)
    Debug.WriteLine(s);
```



We are now ready to connect to the WiFi network.

```
WiFiNetworkInterfaceSettings networkSettings = new WiFiNetworkInterfaceSettings();
networkSettings.Ssid = "YourNetwork";
networkSettings.Password = "YourPassword";
networkSettings.DhcpEnable = true;
networkSettings.DynamicDnsEnable = true;

networkController.SetInterfaceSettings(networkSettings);
networkController.SetAsDefaultController();

//Wait for IP address
var isReady = false;
networkController.NetworkAddressChanged += (a, b) => {
    var ipProperties = a.GetIPProperties();
    Debug.WriteLine("IP: " + ipProperties.Address);
    var address = ipProperties.Address.GetAddressBytes();
    if (address[0] != 0 && address[1] != 0)
        isReady = true;
};

Debug.WriteLine("Enable the network controller.");
networkController.Enable();
Debug.WriteLine("Network controller is enabled.");

while (!isReady)
    Thread.Sleep(100);
```

If the WiFi SSID was not found, or the password was incorrect, the system will raise an exception when enabling the network interface. If and when connected to WiFi, the device will have IP 0.0.0.0 until DHCP has released an IP, which is when the device will receive the new IP event. The output window shows what is happening.

```
Enable the network controller.
Network controller is enabled.
IP: 0.0.0.0
Connection changed!
IP: 192.168.86.45
```

Are we really connected? You can verify by executing a ping command from a PC that is connected to the same network.

```
Pinging 192.168.86.45 with 32 bytes of data:
Reply from 192.168.86.45: bytes=32 time=9ms TTL=255
Reply from 192.168.86.45: bytes=32 time=36ms TTL=255
Reply from 192.168.86.45: bytes=32 time=23ms TTL=255
Reply from 192.168.86.45: bytes=32 time=12ms TTL=255

Ping statistics for 192.168.86.45:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 9ms, Maximum = 36ms, Average = 20ms
```

We know our device is now available on a local network, but does it have an internet connection? We can check by fetching the IP address of a domain name. This will get resolved through DNS, which needs an internet connection. The System.Net needs to be added as now we are using standard .NET networking services.

```
using System.Net;

IPHostEntry ip = Dns.GetHostEntry("GHIElectronics.com");
Debug.WriteLine("GHI Electronics' IP = " + ip.AddressList[0].ToString());
```

The output now should look like:

```
Enable the network controller.
Network controller is enabled.
IP: 0.0.0.0
IP: 192.168.86.45
GHI Electronics' IP = 45.55.96.103
```

Let’s pin ghielectronics.com on the PC to make sure we have the correct IP.

```
Pinging ghielectronics.com [45.55.96.103] with 32 bytes of data:
Reply from 45.55.96.103: bytes=32 time=36ms TTL=46
Reply from 45.55.96.103: bytes=32 time=36ms TTL=46
Reply from 45.55.96.103: bytes=32 time=39ms TTL=46
Reply from 45.55.96.103: bytes=32 time=38ms TTL=46

Ping statistics for 45.55.96.103:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 36ms, Maximum = 39ms, Average = 37ms
```

SOCKETS

Sockets in .NET and point of connections are made between devices. They are available under the System.Net.Sockets namespace. There are thousands of tutorials and articles online showing the use of sockets in .NET. We will only scratch the surface here.

UDP

One kind of socket is User Datagram Protocol (UDP), which is an important member of TCP/IP stack. It is a simple connectionless protocol. That means it sends data from a point to another without establishing a connection and without any error checking. UDP does not know if the receiver had received the data or not. The power of UDP is in its simplicity, making it the base of many other protocols, such as DNS, DHCP, NTP, and TFTP.

For the sake of demonstrating how UDP works, we will build a simple APP on FEZ Bit and another on the PC and let them talk to each other. The FEZ Bit will send “Hello PC” to the PC. You will need to change the code to your PC’s correct IP address. The System.Text needs to be added for Encoding the message.

```
using System.Text;

Socket socket = new Socket(AddressFamily.InterNetwork, SocketType.Dgram, ProtocolType.Udp);
IPAddress DestinationIP = new IPAddress(new byte[] { 192,168,86,44 });
IPEndPoint DestinationEndPoint = new IPEndPoint(DestinationIP, 2000);
String msg = "Hello PC";
byte[] bytesToSend = Encoding.UTF8.GetBytes(msg);
while (true) {
    socket.SendTo(bytesToSend, bytesToSend.Length, SocketFlags.None, DestinationEndPoint);
    while (socket.Poll(2000000, SelectMode.SelectRead)) {
        if (socket.Available > 0) {
            byte[] inBuf = new byte[socket.Available];
            EndPoint recEndPoint = new IPEndPoint(IPAddress.Any, 0);
            socket.ReceiveFrom(inBuf, ref recEndPoint);
            if (!recEndPoint.Equals(DestinationEndPoint))
        continue;
            Debug.WriteLine(new String(Encoding.UTF8.GetChars(inBuf)));
        }
    }
}
```

```
Message From 192.168.86.45
Hello PC
Message From 192.168.86.45
Hello PC
Message From 192.168.86.45
Hello PC
Message From 192.168.86.45
Hello PC
```

When the PC's console app sees the message, it will print it out and then it will reply with *Hello FEZ Bit*. Start a C# console app and add the following code. Note that we are continuing to use top-level-statements.

```
using System;
using System.Text;
using System.Net.Sockets;
using System.Net;

Socket socket = new Socket(AddressFamily.InterNetwork, SocketType.Dgram, ProtocolType.Udp);
IPEndPoint endPoint = new IPEndPoint(IPAddress.Any, 2000);
socket.Bind(endPoint);
while (true) {
    while (socket.Poll(200000, SelectMode.SelectRead)) {
        if (socket.Available > 0) {
            byte[] inBuf = new byte[socket.Available];
            EndPoint recEndPoint = new IPEndPoint(IPAddress.Any, 0);
            socket.ReceiveFrom(inBuf, ref recEndPoint);
            Console.WriteLine("Message From " +
((IPEndPoint)recEndPoint).Address.ToString());
            Console.WriteLine(new string(Encoding.UTF8.GetChars(inBuf)));
            String msg = "Hello FEZ Bit";
            byte[] bytesToSend = Encoding.UTF8.GetBytes(msg);
            socket.SendTo(bytesToSend, bytesToSend.Length, SocketFlags.None,
            (IPEndPoint)recEndPoint);
        }
    }
}
```

```
Hello FEZ Bit
Hello FEZ Bit
Hello FEZ Bit
Hello FEZ Bit
```

TCP

Transmission Control Protocol (TCP) is connection-oriented protocol. The communication starts with two devices connecting and establishing a connection. There is a server side, which listens for the client requesting for a connection. Then data gets transferred between the server and the client. When they are done, the connection is terminated. The TCP protocol also guarantees that data was received, and the data is also received in the right order. Almost everything around the web uses TCP; however, a higher layer protocol is built on top of TCP. Examples can be HTTP (web browsing), FTP (file transfer), SMTP (sending emails) and POP3 (receiving emails).

This example will create a very basic HTTP server and send back a static page to the client.

```
IPHostEntry ip = Dns.GetHostEntry("GHIElectronics.com");
Debug.WriteLine("GHI Electronics' IP = " + ip.AddressList[0].ToString());

Socket server = new Socket(AddressFamily.InterNetwork,
SocketType.Stream, ProtocolType.Tcp);
IPEndPoint localEndPoint = new IPEndPoint(IPAddress.Any, 1200);
server.Bind(localEndPoint);
server.Listen(1);
while (true) {
    // Wait for a client to connect.
    Socket clientSocket = server.Accept();
    // Process the client request. true means asynchronous.
    new ProcessClientRequest(clientSocket, true);
}

class ProcessClientRequest {
    private Socket m_clientSocket;
    public ProcessClientRequest(Socket clientSocket, Boolean asynchronously) {
        m_clientSocket = clientSocket;
        if (asynchronously)
            // Spawn a new thread to handle the request.
            new Thread(ProcessRequest).Start();
        else ProcessRequest();
    }
    private void ProcessRequest() {
        const Int32 c_microsecondsPerSecond = 1000000;
        // 'using' ensures that the client's socket gets closed.
        using (m_clientSocket) {
            // Wait for the client request to start to arrive.
            Byte[] buffer = new Byte[1024];
            if (m_clientSocket.Poll(5 * c_microsecondsPerSecond,
            SelectMode.SelectRead)) {
                // If 0 bytes in buffer, then the connection has been closed,
                // reset, or terminated.
                if (m_clientSocket.Available == 0)
                    return;
                // Read the first chunk of the request (we don't actually do
                // anything with it).
                Int32 bytesRead = m_clientSocket.Receive(buffer,
                m_clientSocket.Available, SocketFlags.None);
```

```
            // Return a static HTML document to the client.

String s =
"HTTP/1.1 200 OK\r\nContent-Type: text/html; charset=utf-8\r\n\r\n<html>"+
"<head><title>TinyCLR OS Web Server on FEZ Bit </title></head>" +
"<body><h1>Learn more about TinyCLR by clicking "+
"<a href=\"http://www.ghielectronics.com/\">here! </a></body></html>";

                byte[] buf = Encoding.UTF8.GetBytes(s);
                int offset = 0;
                int ret = 0;
                int len = buf.Length;
                while (len > 0) {
                    ret = m_clientSocket.Send(buf, offset, len, SocketFlags.None);
                    len -= ret;
                    offset += ret;
                }
                m_clientSocket.Close();
            }
        }
    }
}
```

Open a web browser and type in FEZ Bit's IP address and then add port 1200.

Here is the assigned IP:

```
Network controller is enabled.
IP: 0.0.0.0
IP: 192.168.86.250
```

And here is the browser seeing the page at port 1200:

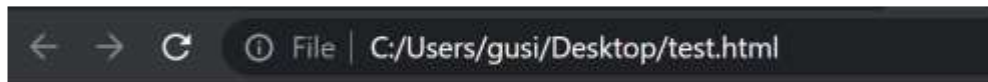
HTTP

We have faked HTTP with the previous TCP example, but TinyCLR includes proper HTTP support. The HTTP library uses TCP sockets internally.

HyperText Transfer Protocol (HTTP) is used to transfer HTML pages from web server to web browsers. If you are not familiar with HTML, open notepad and add this text in it.

```
<html>
<body>

<h1>FEZ Bit</h1>
<p>C# for IoT and Embedded Systems was never easier!</p>

</body>
</html>
```

Now, save the file and change its extension from txt to html. Open the html file in your favorite browser.

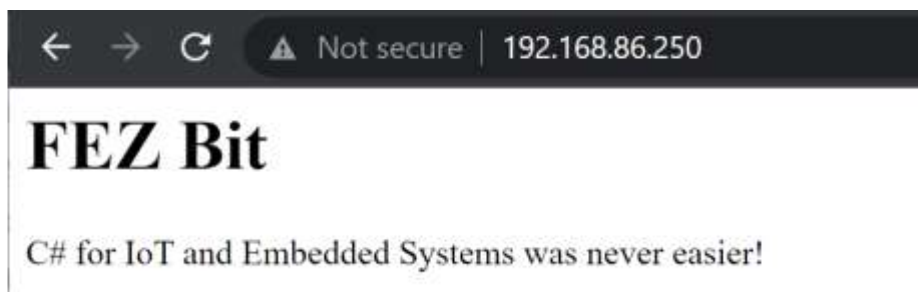


HTML is a way to use readable text to control what the browser shows, from fonts to colors to images. HTTP transfers HTML pages but also handles other tasks as well.

The HTTP library is found under GHIElectronics.TinyCLR.Networking.Http NuGet, but the namespace is the usual System.Net.

```
HttpListener listener = new HttpListener("http", 80);
listener.Start();
while (true) {
    HttpListenerResponse response = null;
    HttpListenerContext context = null;
    try {
        context = listener.GetContext();
        response = context.Response;
        // We are ignoring the request, assuming GET
        // HttpListenerRequest request = context.Request;
        // Sends response:
        response.StatusCode = (int)HttpStatusCode.OK;
        byte[] HTML = Encoding.UTF8.GetBytes(
        "<html><body>" +
        "<h1>FEZ Bit</h1>" +
        "<p>C# for IoT and Embedded Systems was never easier!" +
        "</body></html>");
        response.ContentType = "text/html";
        response.OutputStream.Write(HTML, 0, HTML.Length);
        response.Close();
    } catch {
        if (context != null) {
            context.Close();
        }
    }
}
```

Try it in the browser. Note how we do not need to add the port number since 80 is the default port for HTTP.



Let's up the page a notch and add a button to control the LED. This example will keep an eye on POST requests.

```
HttpListener listener = new HttpListener("http", 80);
listener.Start();
var led = GpioController.GetDefault().OpenPin(FEZBit.GpioPin.Led);
led.SetDriveMode(GpioPinDriveMode.Output);
while (true) {
    HttpListenerResponse response = null;
    HttpListenerContext context = null;
    try {
        context = listener.GetContext();
        response = context.Response;
        // The button is pressed
        if (context.Request.HttpMethod == "POST") {
            if (led.Read() == GpioPinValue.High)
                led.Write(GpioPinValue.Low);
            else
                led.Write(GpioPinValue.High);
        }
        // Sends response
        response.StatusCode = (int)HttpStatusCode.OK;
        byte[] HTML = Encoding.UTF8.GetBytes(
        "<html><body>" +
        "<h1>Hosted on FEZ Bit</h1>" +
        "<p>Click button to toggle the LED.</p>" +
        "<form action=\"\" method=\"post\">" +
        "<input type=\"submit\" value=\"Toggle!\">" +
        "</form>" +
        "</body></html>");
        response.ContentType = "text/html";
        response.OutputStream.Write(HTML, 0, HTML.Length);
        response.Close();
    } catch {
        if (context != null) {
            context.Close();
        }
    }
}
```

With this example, we are now controlling a physical device from a webpage.

Try the same from a phone that is connected to the same network. Pretty cool, right?! With a public IP address, you can control the LED from anywhere in the world.

Can you modify the code to set the buzzer frequency from a webpage?

TELNET

Telnet is a very simple but very useful TCP connection between 2 nodes. Most terminal software programs that work with serial ports also work with Telnet network connections. There are countless Telnet applications, for PC and for mobile devices. We will be using Tera Term for the PC .

FEZ Bit will be the server, waiting for client requests (Tera Term) to handle specific tasks. We need to run a listening socket and wait for a connection. A new thread is created for every connection, so you can have more than one connection!

```
var led = GpioController.GetDefault().OpenPin(FEZBit.GpioPin.Led);
led.SetDriveMode(GpioPinDriveMode.Output);
Socket server = new Socket(AddressFamily.InterNetwork, SocketType.Stream,
ProtocolType.Tcp);
// Telnet usually uses port 23
IPEndPoint localEndPoint = new IPEndPoint(IPAddress.Any, 23);
server.Bind(localEndPoint);
// start listening
server.Listen(1);
while (true) {
    // Wait for a client
    Socket Sock = server.Accept();
    new TelnetProcess(Sock, led);
}

class TelnetProcess {
    byte[] prompt = Encoding.UTF8.GetBytes("\r\nFEZ >");
    private Socket clientSocket;
    bool EchoIsEnabeled = true;
    GpioPin _led;
    public TelnetProcess(Socket Sock, GpioPin ledPin) {
        clientSocket = Sock;
        _led = ledPin;
        // Spawn a new thread
        new Thread(Process).Start();
    }
    private void HandleCommand(string cmd) {
        string str;
        cmd = cmd.ToUpper();
        switch (cmd) {
            case "ECHO":
                EchoIsEnabeled = !EchoIsEnabeled;
                clientSocket.Send(prompt);
                str = "EchoIsEnabeled = " + EchoIsEnabeled;
                clientSocket.Send(Encoding.UTF8.GetBytes(str), 0, str.Length,
                SocketFlags.None);
                break;
            case "LED ON":
                _led.Write(GpioPinValue.High);
                clientSocket.Send(prompt);
                str = "LED is now on";
                clientSocket.Send(Encoding.UTF8.GetBytes(str), 0, str.Length,
                SocketFlags.None);
                break;
            case "LED OFF":
                _led.Write(GpioPinValue.Low);
                clientSocket.Send(prompt);
                str = "LED is now off";
```

```
                clientSocket.Send(Encoding.UTF8.GetBytes(str), 0, str.Length,
                SocketFlags.None);
                break;
        }
    }
    private void Process() {
        byte[] command = new byte[100];
        int index = 0;
        using (clientSocket) {
            clientSocket.Send(prompt);
            while (true) {
                if (clientSocket.Receive(command, index, 1, SocketFlags.None) == 0) {
                    clientSocket.Close();
                    break;
                }
                if (command[index] == '\r') {
                    // do we have some data?
                    string cmd = new string(Encoding.UTF8.GetChars(command), 0, index);
                    HandleCommand(cmd);
                    index = 0;
                    clientSocket.Send(prompt);
                }
                else if (command[index] >= 32 && command[index] <= 126) {
                    // Echo back
                    if (EchoIsEnabeled)
                        clientSocket.Send(command, index, 1, SocketFlags.None);
                    index++;
                    if (index >= command.Length) {
                        Debug.WriteLine("We have too much data!");
                        index = 0;//dump it all
                    }
                }
            }
        }
    }
}
```

Note that terminals do not echo (show) what you type in them. What you typed is simply sent to FEZ Bit. We have added a little more code to echo back what was entered when enter key is pressed. There is an option to enable local echo as well, simply browse around the terminal settings later.

Open Tera Term and enter the FEZ Bit’s IP address.

Once the connection is established, FEZ Bit responds with “FEZ >” prompt.

Start typing in the command. Remember that you may not see what you type in. So just type it and hit enter.

Try **led on** and hit the enter key. The LED will come on and FEZ Bit will respond with **LED is now on**. Now, try **led off**.

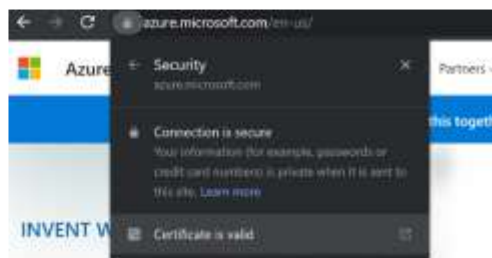
TLS

Transport Layer Security (TLS) is a security layer that secures the data transferred between server and client. It also includes authentication so a client and server can verify that the device on the other side is the true device they are trying to reach and not some fake pirate.

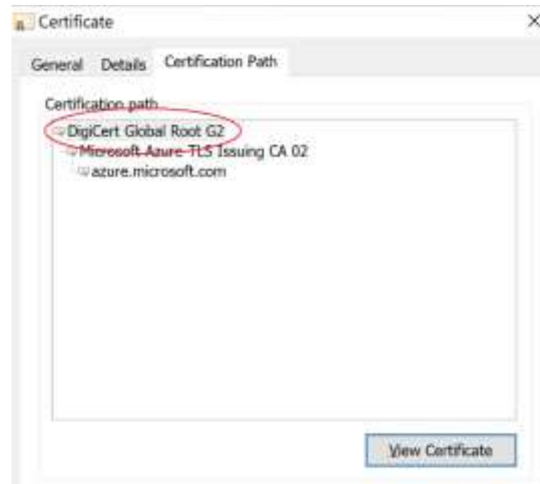
TinyCLR handles TLS in its internal secure memory making it extremely difficult for unwanted parties to "see" the transferred data. Both, TLS client and TLS server can be implanted within TinyCLR.

More info is located at https://docs.ghielectronics.com/software/tinyclr/tutorials/tls.html .

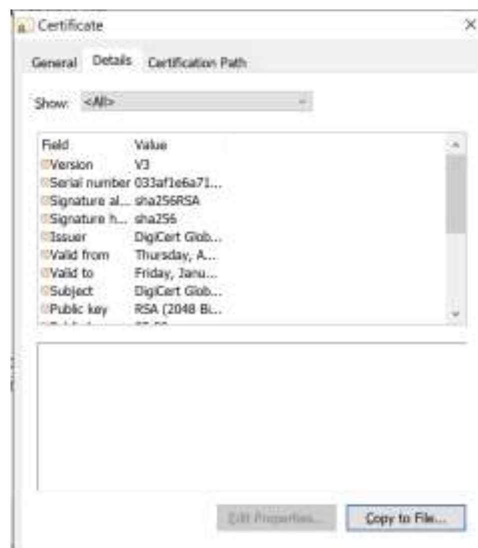
For TLS security to work properly, FEZ Bit needs the root certificate that a server uses to authenticate its connection. Your PC comes preloaded with trusted root certificates, and one of them is the one the server uses. Open a browser and visit the same website you want to connect to from FEZ Bit, for example azure.com. Using Edge or Chrome, you will find a lock icon by the website. Click on it and then find the option to open the certificate manager, which is *Connection is secure->Certificate is valid* on Chrome.



From the certificate path, we can see the root certificate, the one all the way on top. Select it and click **View Certificate**.



And now from under details, click **Copy to File…**



You can save the certificate as DER or Base-64. We are going to use DER. This saved certificate needs to be added as a resource. If you are not sure how, see the Resources chapter. You'll need to add 'using System.Security.Cryptography.X509Certificates' statement to use the certificate. The certificate is loaded from the project resources as follows:

```
var CaCert = new
X509Certificate(test.Resources.GetBytes(test.Resources.BinaryResources.azure_root));
```

MQTT

MQTT is a light-weight messaging protocol for sensors that is supported by all major cloud services. We will be using MQTT with the Cloud Services chapter.

CLOUD SERVICES

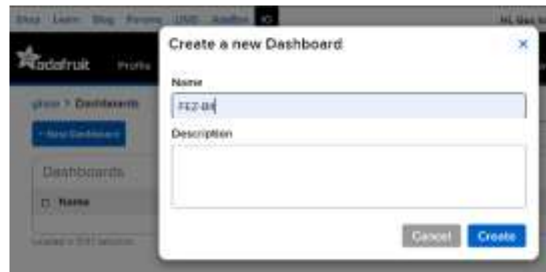
FEZ Bit can connect to any cloud service securely, thanks to the full featured support of TinyCLR. Azure, AWS, Google Cloud, and others work beautifully with TinyCLR. Most cloud services use MQTT, which is built in TinyCLR.

This section assumes that you have an active WiFi connection, and your device has received an IP address. The code and the details are found in the Networking chapter. The code provided here is an addition to the WiFi setup code.

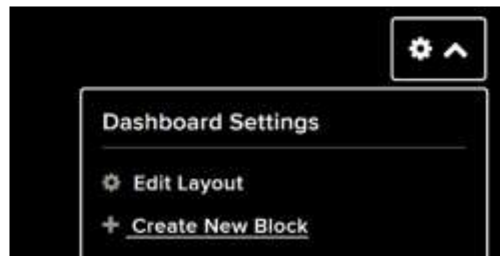
ADAFRUIT IO

While Adafruit IO is more limited than the major players in cloud services, it is very easy to use and there's a free option making this a great way to test IoT proof-of-concepts and prototypes.

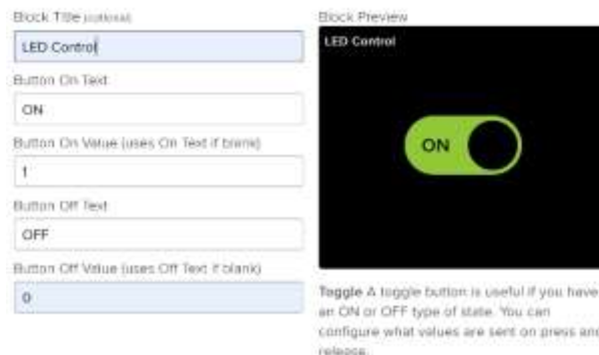
Start by making an account and logging in. Then create a new dashboard, calling it FEZ-Bit.



We can now start adding blocks to our dashboard.



Let's start by adding ToggleSwitch. In the block settings window, set the *on* value to 1 and *off* value to 0.



And that is all we need! We now have a toggle button that sends an event when it is clicked. It will send 1 when on and 0 when off. The feed for this toggle switch can be found from under feeds in the top menu. From the ToggleSwitch feed we can now click on feed info. There, we can find the MQTT feed.

Name

ToggleSwitch

Maximum length: 128 characters. Used: 12

Key

toggleswitch

Changing the key will change API URLs and MQTT subscription topics. The only characters we permit are lower case english letters ("a" to "z"), numbers, and dash ("-").

See our guide to naming things in Adafruit IO for more information about how we handle the formatting of **names** and **keys**.

Current Endpoints

We will use MQTT to access Adafruit IO and so we will need GHIElectronics.TinyCLR.Networking.Mqtt NuGet/namespace. And do not forget to download the **root certificate** required for a secure TLS connection. The steps to download and add the root certificate was explained earlier in the TLS section.

```
// Change to match your setup
var dashboardId = "fez-bit";
var username = "gissa";
var key = "MySecretAdafruitKey";
var feed = "gissa/feeds/toggleswitch";

// need the proper root certificate, see TLS section
var CaCert = new
X509Certificate(test.Resource.GetBytes(test.Resource.BinaryResources.adafruit_root));

var mqttHost = "io.adafruit.com";
var mqttPort = 8883;
var led = GpioController.GetDefault().OpenPin(FEZBit.GpioPin.Led);
led.SetDriveMode(GpioPinDriveMode.Output);
var clientSetting = new MqttClientSetting {
    BrokerName = mqttHost,
    BrokerPort = mqttPort,
    ClientCertificate = null,
    CaCertificate = CaCert,
    SslProtocol = System.Security.Authentication.SslProtocols.Tls12,
};
var client = new Mqtt(clientSetting);
var connectSetting = new MqttConnectionSetting {
    ClientId = dashboardId,
    UserName = username,
    Password = key
};
var returnCode = client.Connect(connectSetting);
if (returnCode == ConnectReturnCode.ConnectionAccepted)
    Debug.WriteLine("Connected to Adafruit IO");
var packetId = 1;
// Subscribe to a feed
client.Subscribe(new string[] { feed }, new QoSLevel[] { QoSLevel.ExactlyOnce },
    (ushort)packetId++);
client.PublishReceivedChanged += Client_PublishReceivedChanged;

Thread.Sleep(Timeout.Infinite);

void Client_PublishReceivedChanged(object sender, string topic, byte[] data, bool duplicate,
QoSLevel qosLevel, bool retain) {
    if (data[0] == '1')
        led.Write(GpioPinValue.High);
    else
        led.Write(GpioPinValue.Low);
}
```

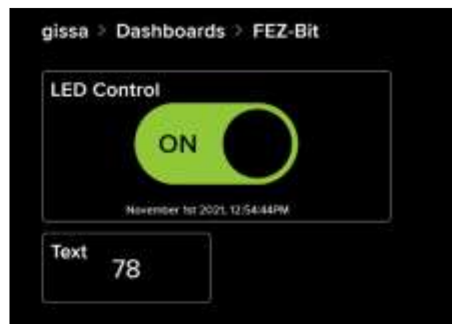
Once FEZ Bit subscribes to the MQTT feed, an event will be raised with every switch click. The event will be '1' when the button is ON and '0' when the button is OFF. These are the values we have specified when we have configured the switch. Those values can then be used, like to control the LED… or a garage door!

The previous example subscribed to a feed to receive data. We can also publish to a feed. We will add a Text block that gets its data (text) from a feed that is called *text*.

Only few lines of code are needed to send something to the text block. In this example, we are sending an increasing number that starts at 50. Replace the **Thread.Sleep(Timeout.Infinite);** line with the code below.

```
var x = 50;
while (true) {
    x++;
    // Publish a topic
    client.Publish("gissa/feeds/text", Encoding.UTF8.GetBytes(x.ToString()),
QoSLevel.MostOnce, false, (ushort)packetId);
    Thread.Sleep(2000);
}
```

The text block will show the changing number.



MICROSOFT AZURE

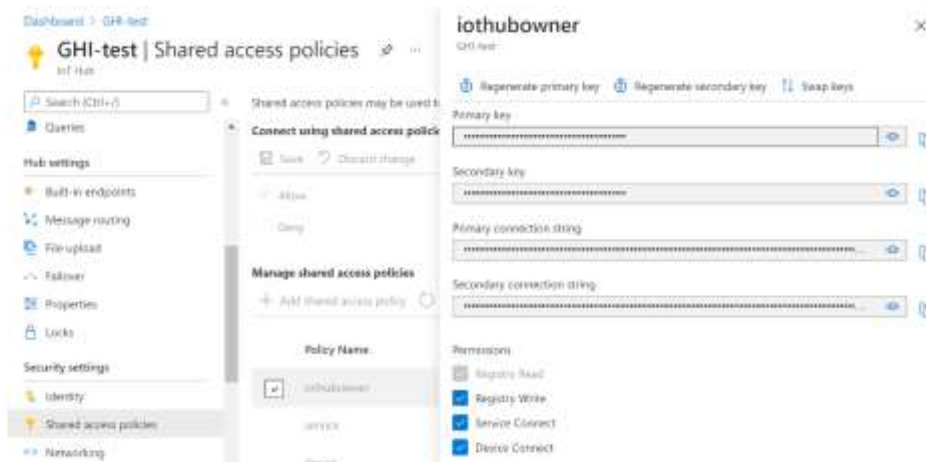
There are multiple services from Azure than can be used with TinyCLR and SITCore. However, most users interested in IoT will end up using IoT hub. The free tier is plenty to try this example out.

Start by logging into Azure portal and creating a resource.

We have ours called *GHI-test*.

This IoT Hub can contain all the IoT devices you need to communicate with. Let's add a new device that is called *FEZ-Bit*. We will select Symmetric Key authentication option.

IoT Hub is now ready, and we are ready to code FEZ Bit to talk to it. FEZ Bit needs to know our IoT Hub name and key and then it also needs to know the device name. The IoT Hub key is found under Shared access policies. In the demo we are using the **iothubowner**. Select that and copy the **Primary Key**.



You will need this key to replace "MySecretKey" in the code below.

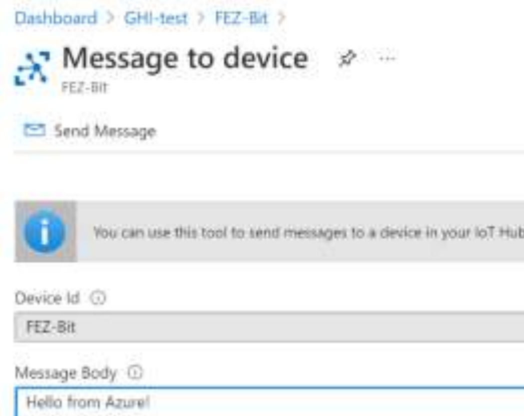
The last step is to load the root certificate. The TLS section shows how to find and save the certificate into resources.

To access IoT hub we need GHIElectronics.TinyCLR.Drivers.Azure.SAS and GHIElectronics.TinyCLR.Networking.Mqtt NuGet/namespace.

```
var caCert = new
X509Certificate(test.Resource.GetBytes(test.Resource.BinaryResources.azure_root));
// Config IoT Hub
var iotHubName = "GHI-test.azure-devices.net";
const string iotHubKey = MySecretKey;
const string deviceId = "FEZ-Bit";
// Data time is important to calculate expire time
SystemTime.SetTime(new DateTime(2021, 11, 1));
//----------------------------------------

var iotHubPort = 8883;
```

```
var username = string.Format("{0}/{1}", iotHubName, deviceId);
var sas = new SharedAccessSignatureBuilder() {
    Key = iotHubKey,
    KeyName = "iothubowner",
    Target = iotHubName,
    TimeToLive = TimeSpan.FromDays(1) // at least 1 day.
};
var topicDeviceToServer =
string.Format("devices/{0}/messages/events/", deviceId);
var topicService2Device =
string.Format("devices/{0}/messages/devicebound/#", deviceId);

var clientSetting = new MqttClientSetting {
    BrokerName = iotHubName,
    BrokerPort = iotHubPort,
    ClientCertificate = null,
    CaCertificate = caCert,
    SslProtocol = System.Security.Authentication.SslProtocols.Tls12
};
var client = new Mqtt(clientSetting);
client.PublishReceivedChanged += (p1, p2, p3, p4, p5, p6) => {
    Debug.WriteLine("Received message: " + Encoding.UTF8.GetString(p3));
};
var connectSetting = new MqttConnectionSetting {
    ClientId = deviceId,
    UserName = username,
    Password = sas.ToSignature()
};
var returnCode = client.Connect(connectSetting);
if (returnCode != ConnectReturnCode.ConnectionAccepted)
    throw new Exception("Could not connect!");

ushort packetId = 1;
client.Subscribe(new string[] { topicService2Device }, new QoSLevel[]
    { QoSLevel.ExactlyOnce }, packetId++);
client.Subscribe(new string[] { topicDeviceToServer }, new QoSLevel[]
    { QoSLevel.ExactlyOnce }, packetId++);

var btn = GpioController.GetDefault().OpenPin(FEZBit.GpioPin.ButtonA);
btn.SetDriveMode(GpioPinDriveMode.InputPullUp);
btn.ValueChangedEdge = GpioPinEdge.FallingEdge;
btn.ValueChanged += (GpioPin sender, GpioPinValueChangedEventArgs e) => {
    client.Publish(topicDeviceToServer, Encoding.UTF8.GetBytes
        ("Your message"), QoSLevel.MostOnce, false, packetId++);
};

Thread.Sleep(Timeout.Infinite);
```

The Azure portal can be used to send messages to the device.



And the message sent will show in the output window.

Pushing button-A will also send a message to the cloud. Unfortunately, there is no way to see the message in the portal itself. You will need software to do that, such as Microsoft's Azure IoT Explorer https://github.com/Azure/azure-iot-explorer/releases .

Of course, the power of Azure comes in combining multiple Azure services, like pipe the data from IoT hub into analytics and beyond.

OTHER CLOUD SERVICES

With having MQTT and HTTP in TinyCLR, a SITCore device such as FEZ Bit can connect to any cloud service, such as Amazon Web Services (AWS) and Google cloud.

https://docs.ghielectronics.com/software/tinyclr/tutorials/google-cloud.html

https://docs.ghielectronics.com/software/tinyclr/tutorials/aws.html

CRYPTOGRAPHY

Cryptography has been an important part of technology and SITCore (Secure IoT Core). SITCore is not complete without crypto support.

XTEA

XTEA, with its 16byte (128bit) key, is a popular algorithm that has a decent level of security, and it is simple and fast. XTEA is not suitable for banks and critical data.

There are two important things to keep in mind when working with XTEA. First, the key is always 16Bytes, usually represented as 4x 32bit values. Second, the data must be 8Byte or multiple of 8Byte in size.

The XTEA library is found in the GHIElectronics.TinyCLR.Core and GHIElectronics.TinyCLR.Cryptography NuGet/namespace.

```
using System.Text;
using GHIElectronics.TinyCLR.Cryptography;
using System.Diagnostics;

var crypto = new Xtea(new uint[] { 0x01234567, 0x89ABCDEF, 0xFEDCBA98, 0x76543210 });

byte[] dataToEncrypt = Encoding.UTF8.GetBytes("Data to encrypt.");
byte[] encryptedData;
byte[] decryptedData;

//Data must be a multiple of 8 bytes.
encryptedData = crypto.Encrypt(dataToEncrypt, 0, (uint)dataToEncrypt.Length);
decryptedData = crypto.Decrypt(encryptedData, 0, (uint)encryptedData.Length);

Debug.WriteLine("Decrypted: " + Encoding.UTF8.GetString(decryptedData));
```

XTEA has probably been implemented in almost every known coding language! Wikipedia also includes a C implementation https://en.wikipedia.org/wiki/XTEA .

RSA

XTEA uses the same key to encode and decode data. This symmetrical key is a problem in some cases. You will need to share your key with the receiver for them to decode the data. But this also means they can use that key to decode new data. RSA is asymmetrical key pair. If you want data from someone, but they only want you to be able to see the data, you can send them the public key that is used by them to decode the data and send to you. While anyone else can see the data and can also see the public key, they have no way of decoding the data. Only you can because you have the private key pair of that public key. You, and only you, can decode that data, within reason!

The RSA implementation is a subset of the full .NET and found in the GHIElectronics.TinyCLR.Cryptography NuGet/namespace.

```
using System.Text;
using GHIElectronics.TinyCLR.Cryptography;
using System.Diagnostics;

byte[] dataToEncrypt = Encoding.UTF8.GetBytes("Data to Encrypt");
byte[] encryptedData;
byte[] decryptedData;

using (RSACryptoServiceProvider RSA = new RSACryptoServiceProvider(2048)) {
    //Encrypt data.
    using (RSACryptoServiceProvider encryptRSA = new RSACryptoServiceProvider()) {

        encryptRSA.ImportParameters(RSA.ExportParameters(false));
        encryptedData = encryptRSA.Encrypt(dataToEncrypt);
    }

    //Decrypt data.
    using (RSACryptoServiceProvider decryptRSA = new RSACryptoServiceProvider()) {

        decryptRSA.ImportParameters(RSA.ExportParameters(true));
        decryptedData = decryptRSA.Decrypt(encryptedData);
    }
}

Debug.WriteLine("Decrypted: " + Encoding.UTF8.GetString(decryptedData));
```

Keys are handled by ExportParameters and ImportParameters methods. A random key pair is automatically generated by the service provider.

FILE SYSTEM

File system is supported in TinyCLR. The implementation is a subset of full .NET. The internal drivers fully support FAT16 or FAT32 file systems, with no limitations beyond the FAT file system itself! This book is more about IoT and robotics and this section only covers the basics to get you started.

There is also Tiny File System that is used with memory chips. We will not cover Tiny File System here, but you can find more info on TinyCLR Docs website: https://docs.ghielectronics.com/software/tinyclr/tutorials/file-system.html .

SD CARDS

FEZ Bit includes a micro SD card connector. Plug in a micro SD card and try the following code that prints all file names found in the root directory.

The file system libraries are found in the GHIElectronics.TinyCLR.IO NuGet/namespace and the SD Storage drivers are found in GHIElectronics.TinyCLR.Devices.Storage NuGet/namespace. We will also need System.IO namespace.

```
using System.IO;
using GHIElectronics.TinyCLR.IO;
using GHIElectronics.TinyCLR.Devices.Storage;
using GHIElectronics.TinyCLR.Pins;


var sd = StorageController.FromName(SC20100.StorageController.SdCard);
var drive = FileSystem.Mount(sd.Hdc);

var directory = new DirectoryInfo(drive.Name);
var files = directory.GetFiles();

foreach (var f in files) {
    System.Diagnostics.Debug.WriteLine(f.Name);
}
```

Accessing files is like how it is done on the PC. This example will open or create a file called *Test.txt* and then write the current time and date to it.

```
using System;
using System.Text;
using System.IO;
using GHIElectronics.TinyCLR.IO;
using GHIElectronics.TinyCLR.Devices.Storage;
using GHIElectronics.TinyCLR.Pins;


var sd = StorageController.FromName(SC20100.StorageController.SdCard);
var drive = FileSystem.Mount(sd.Hdc);
```

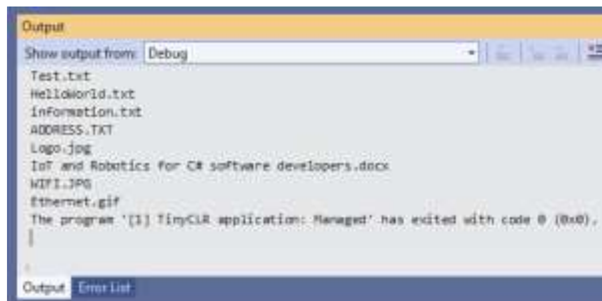
```
var file = new FileStream($@"{drive.Name}Test.txt", FileMode.OpenOrCreate);
var bytes = Encoding.UTF8.GetBytes(DateTime.UtcNow.ToString() +
    Environment.NewLine);

file.Write(bytes, 0, bytes.Length);

file.Flush();
FileSystem.Flush(sd.Hdc);
```

USB MASS STORAGE

USB Host is supported by TinyCLR and the signals are found on FEZ Bit, however no connector is mounted on it. See the USB Host section for further details.

Files are handled on USB in the same way it is done with SD. The only difference is that the storage controller is now USB instead of SD. The USB Host libraries are in GHIElectronics.TinyCLR.Devices.UsbHost NuGet/namespace.

The code to read a list of files is the same as before, but there is additional code to detect that a device has been inserted and it is a mass storage device.




```
using System;
using System.Diagnostics;
using System.Text;
using System.IO;
using System.Threading;
using GHIElectronics.TinyCLR.IO;
using GHIElectronics.TinyCLR.Devices.Storage;
using GHIElectronics.TinyCLR.Devices.UsbHost;
using GHIElectronics.TinyCLR.Pins;

var usbHostController = UsbHostController.GetDefault();

usbHostController.OnConnectionChangedEvent +=
    UsbHostController_OnConnectionChangedEvent;

usbHostController.Enable();

Thread.Sleep(Timeout.Infinite);

void UsbHostController_OnConnectionChangedEvent
        (UsbHostController sender, DeviceConnectionEventArgs e) {

    switch (e.DeviceStatus) {
        case DeviceConnectionStatus.Connected:
            switch (e.Type) {
                case BaseDevice.DeviceType.MassStorage:
                    var storageController = StorageController.FromName
                        (SC20260.StorageController.UsbHostMassStorage);
```

```
                    var drive = FileSystem.Mount(storageController.Hdc);
                    var driveInfo = new DriveInfo(drive.Name);

                    var directory = new DirectoryInfo(drive.Name);
                    var files = directory.GetFiles();

                    foreach (var f in files) {
                        System.Diagnostics.Debug.WriteLine(f.Name);
                    }

                    break;
            }
            break;

        case DeviceConnectionStatus.Disconnected:
            Debug.WriteLine("Device Disconnected");
            //Unmount file system if it was mounted.
            break;

    }
}
```

FILE SYSTEM CONSIDERATIONS

FAT File System is only capable of FAT32 and FA16. A media formatted as FAT12 will not work. The file system does a lot of data buffering internally to speed up the file access time and to increase the life of the flash media. When you write data to a file, the data is buffered first and then it is written on the media. To make sure the data is stored on the media, we need to “flush” the data. Flushing or closing a file is the only way to guarantee that the data you are trying to write are now on the actual media. In the earlier file write example, we have 2 lines at the end to flush the file and flush the file system.

```
file.Flush();
FileSystem.Flush(sd.Hdc);
```

The file flush will guarantee the file data itself is saved, but the media itself may have some buffering as well. You need to flush both to guarantee there are no buffered data.

Typically, you will write data to the card and flush the file as desired and only flush the media before it is inserted.

Note that the FAT file system is not a journaling file system, and so removing a media before all data is flushed will corrupt the data on the media. This is the same case as on the PC, except the PC regularly flushes data in the background for you.

TIME SERVICES

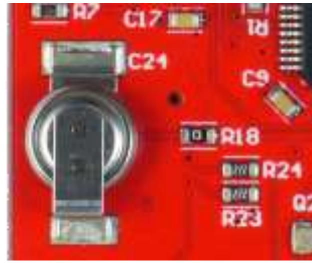
In computer systems, time can mean 2 things. The system time, which is the processor's ticks, is used to handle timing and to manage the system. There is also Real Time Clock (RTC), which is the clock we use for minutes, hours and even dates.

REAL TIME CLOCK

What is DateTime.Now? On the PC, it is the current time. But how did the PC know the current time initially? Reading DateTime.Now on FEZ Bit will result with a random value, which is expected.

With an Internet connection, the system can fetch the time online. But with no internet connection, the system needs to know the current time internally. This is where the Real Time Clock (RTC) comes in handy. This is a circuit that runs on its own power source, usually a small battery on the PC's motherboard. The RTC will keep running even when the PC is powered off. When the PC powers up, it reads the RTC to set the system's internal clock and then from that point on the system does not need the RTC until the next power cycle.

FEZ Bit includes RTC that runs independently using a supercap, which acts like a small battery. To use the RTC, we need to enable the supercap charging circuit.



The RTC drivers are found in GHIElectronics.TinyCLR.Devices.Rtc NuGet/namespace.

```
using GHIElectronics.TinyCLR.Devices.Rtc;

var rtc = RtcController.GetDefault();
rtc.SetChargeMode(GHIElectronics.TinyCLR.Devices.Rtc.BatteryChargeMode.Fast);
```

With an enabled charging circuit, the supercap will be powering up the RTC even when the system main power is off.

Note that reading the RTC may raise an exception if the RTC has a bad time, like the hour is 0 or month is 60! Before we read the RTC time, we should check if it is valid, or we can catch the exception.

```
using System;
using System.Threading;
using System.Diagnostics;
using GHIElectronics.TinyCLR.Devices.Rtc;

var rtc = RtcController.GetDefault();
rtc.SetChargeMode(GHIElectronics.TinyCLR.Devices.Rtc.BatteryChargeMode.Fast);

if(rtc.IsValid == false) {
    Debug.WriteLine("RTC is invalid");
    Thread.Sleep(Timeout.Infinite);
}
while (true) {
    Debug.WriteLine("Current Time     : " + DateTime.Now);
    Debug.WriteLine("Current RTC Time: " + rtc.Now);
    Thread.Sleep(1000);
}
```

Your program will normally have a user interface for the user to set the RTC. Or the program can use the internet to set the RTC. For the sake of this demo, we will just set it to January 1st, 2022 at 11:11:11.

```
using System;
using System.Threading;
using System.Diagnostics;
using GHIElectronics.TinyCLR.Devices.Rtc;
using GHIElectronics.TinyCLR.Native;

var rtc = RtcController.GetDefault();
rtc.SetChargeMode(GHIElectronics.TinyCLR.Devices.Rtc.BatteryChargeMode.Fast);

if (rtc.IsValid) {
    Debug.WriteLine("RTC is Valid");
    // RTC is good so let's use it
    SystemTime.SetTime(rtc.Now);
}
else {
    Debug.WriteLine("RTC is Invalid");
    // RTC is not valid. Get user input to set it
    // This example will simply set it to January 1st 2022 at 11:11:11
    var MyTime = new DateTime(2022, 1, 1, 11, 11, 11);
    rtc.Now = MyTime;
    SystemTime.SetTime(MyTime);
}
while (true) {
    Debug.WriteLine("Current Time     : " + DateTime.Now);
    Debug.WriteLine("Current RTC Time: " + rtc.Now);
    Thread.Sleep(1000);
}
```

Running the code on an invalid RTC will result in this in the output window.

```
The thread '<No Name>' (0x2) has exited with code 0 (0x0).
RTC is Invalid
Current Time     : 01/01/2022 11:11:11
Current RTC Time: 01/01/2022 11:11:11
Current Time     : 01/01/2022 11:11:12
Current RTC Time: 01/01/2022 11:11:12
```

Now, power off the FEZ Bit and give it few seconds/minutes. Now run the program again. We can see that the time is now has advanced and it is no longer 11:11:11.

```
The thread '<No Name>' (0x2) has exited with code 0 (0x0).
RTC is Valid
Current Time     : 01/01/2022 11:16:21
Current RTC Time: 01/01/2022 11:16:21
Current Time     : 01/01/2022 11:16:22
Current RTC Time: 01/01/2022 11:16:22
```

Of course, the system time and RTC time are the same, as we have set the system time on power up to match the RTC. For the sake of demonstration, let us run a program that just reads the time as a default value and shows the system time vs RTC time.

```
Current Time     : 01/01/2017 00:00:11
Current RTC Time: 01/01/2022 11:19:07
Current Time     : 01/01/2017 00:00:12
Current RTC Time: 01/01/2022 11:19:08
Current Time     : 01/01/2017 00:00:13
Current RTC Time: 01/01/2022 11:19:09
```

The system time is starting from January 1st 2017, which is the starting point of time in TinyCLR! The RTC is still running. By the way, can you tell how long it took us to try these demos? We started at 11:11:11 and last example was at 11:19:07.

TIMERS

TinyCLR includes timers that can be used to invoke a method at a specific period and a specific start delay. This example will start invoking Ticker after three seconds and then will keep calling it once a second.

```
using System.Threading;
using System.Diagnostics;

void Ticker(object o) {
    Debug.WriteLine((string)o);
}

Debug.WriteLine("Started...");
Timer timer = new Timer(Ticker, "Hello", 3000, 1000);

Thread.Sleep(Timeout.Infinite);
```

But why not just have a thread that does the same thing?

```
using System.Threading;
using System.Diagnostics;

void Ticker() {
    Thread.Sleep(3000);
    while (true) {
        Debug.WriteLine("Hello");
        Thread.Sleep(1000);
    }

}

Debug.WriteLine("Started...");
new Thread(Ticker).Start();

Thread.Sleep(Timeout.Infinite);
```

The difference is that a thread that contains a one second sleep will sleep for that one second in addition to the time used by other code in the thread. So, if a thread needs 0.5 second to complete what it is doing, sleeping for one second will cause the thread to execute every 1.5 seconds. This also gets more complex as a thread's code can have a different processing time in every loop.

A timer set to invoke a method every second will do so every second regardless of how long the invoked method takes to execute. However, care must be taken because if a timer calls a method every 100 milliseconds, but the method needs more than 100 milliseconds to execute, you will end up flooding the system. The best practice is for timers to invoke methods that execute in a very short time.

USB CLIENT

The USB Client connector is what we are using to connect to a PC and deploy/debug applications. This USB interface can be used for other purposes; however, to do this we need to free it up from it being used as a debug interface. The easiest way is to switch to serial debugging, but the needed pins/interfaces are not available on FEZ Bit. Do not worry, we can simply disable the debugging interface altogether. By the way, disabling the debug interface is one of the many security features built in SITCore and TinyCLR OS.

An important note to keep in mind is that once the interface is disabled, you can no longer debug/deploy to your device. You can still reset the device, however, and we will show you how.

The code to disable the interface is here, but DO NOT RUN IT YET!

```
GHIElectronics.TinyCLR.Native.DeviceInformation.SetDebugInterface(DebugInterface.Disable);
```

Running the code above will set flags in the device, so the device will not have a debug interface on the next restart. Once you run the above code and reset the device, the PC will not see anything on the USB interface. This is bad if you want to debug, but it is good if you want to use the USB interface from within code.

Ok good! We know how to kill debugging! Now, let’s make the USB port do something fun. How about turning it into a mouse that rotates the cursor on the screen.

The USB Client libraries are found in the GHIElectronics.TinyCLR.Devices.UsbClient NuGet/namespace.

```
using GHIElectronics.TinyCLR.Devices.UsbClient;
using GHIElectronics.TinyCLR.Native;
using System;
using System.Threading;

DeviceInformation.SetDebugInterface(DebugInterface.Disable);

var usbclientController = UsbClientController.GetDefault();
var absolutePosition = true;
var mouse = new Mouse(usbclientController, absolutePosition);

mouse.DeviceStateChanged += UsbClientDeviceStateChanged;
mouse.Enable();

Thread.Sleep(Timeout.Infinite);

void UsbClientDeviceStateChanged(RawDevice sender, DeviceState state) {
    var i = 0.0;
    if (state == DeviceState.Configured) {
        new Thread(() => {
            while (true) {
                ((Mouse)sender).MoveCursor(
                    Mouse.MaxRange / 4 + (int)(Math.Cos(i) * Mouse.MaxRange / 10),
                    Mouse.MaxRange / 4 + (int)(Math.Sin(i) * Mouse.MaxRange / 10));
                i += 0.03;
                Thread.Sleep(50);
            }
        }).Start();
    }
}
```

Run the program and nothing will happen the first time it runs. Why? The first program sets the flag to disable the debug interface after reset, but the interface is still active for that run. The next lines will initialize the USB Client interface to simulate a USB mouse. This is where the program will raise an exception because the USB port is used by the debugger.

On the second run, USB is completely free and the first line that disables the interface has no effect, as was done already in the previous run. And now USB is free for the USB code to work fine.

You should now see the cursor on your PC going in a circle! We will open Paint and push left-click to draw… you can now see it better.

But how do we reset the device to load a new program? Disabling the debug interface only stops the debugger in TinyCLR and has nothing to do with the boot loader. We can go back to our earlier steps when we updated the firmware.

1. Press and hold down the left button (LDR) and reset the board while holding LDR down.
2. Open TinyCLR Config tool and select the COMx (GHI Interface) port and click the connect button.
3. From the top menu: Erase -> Erase All

Your device is now blank as if it is brand new. The next step is to load the TinyCLR firmware:

1. Connect to the device, as you just did!
2. Select the appropriate firmware. If you do not have it, download the latest RTW SC20xxx firmware release from https://docs.ghielectronics.com/software/tinyclr/downloads.html .
3. Click the Update Firmware button.

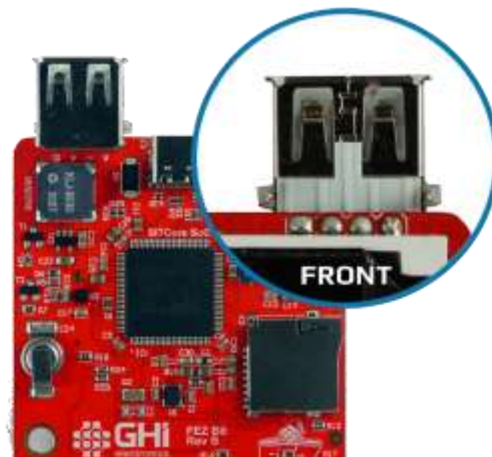
And now you are back to a fully functional device. If you need to use the USB Client extensively and need to debug code over serial port while using USB Client in your application, we suggest using one of the SITCore Development boards.

USB HOST

The USB host is the system that connects to multiple USB devices/clients. For example, the PC is a USB host and it can connect to multiple USB devices like mice, keyboards and mass storage devices.

USB Host is supported by TinyCLR and the signals are found on FEZ Bit, however, no connector is mounted on it. You will need to add a connector or use a different FEZ board, like FEZ Duino, for example.

Using a board that has USB Host built in is the preferred method, but to add a connector to FEZ Bit, you will need a through-hole USB type A connector. The connector needs to be soldered on the holes found on the very top of the board.



Note that USB Host will require 5V to run, and if the FEZ Bit is powered from the accessory, then it will only have 3.3V and USB Host will not work. In Other words, the USB host connector will only work if there is USB power coming on the USB Client connector.

We will not get into deep details as we are assuming the reader here will not be soldering. This example shows a range of supported devices and how they are used.

The USB Host libraries are found in GHIElectronics.TinyCLR.Devices.UsbHost NuGet/namespace.

```
using System.Diagnostics;
using System.Threading;
using GHIElectronics.TinyCLR.Devices.UsbHost;
using GHIElectronics.TinyCLR.Devices.UsbHost.Descriptors;


var usbHostController = UsbHostController.GetDefault();

usbHostController.OnConnectionChangedEvent +=
    UsbHostController_OnConnectionChangedEvent;

usbHostController.Enable();

Thread.Sleep(Timeout.Infinite);


void UsbHostController_OnConnectionChangedEvent
    (UsbHostController sender, DeviceConnectionEventArgs e) {

    Debug.WriteLine("e.Id = " + e.Id + " \n");
    Debug.WriteLine("e.InterfaceIndex = " + e.InterfaceIndex + " \n");
    Debug.WriteLine("e.PortNumber = " + e.PortNumber);
    Debug.WriteLine("e.Type = " + ((object)(e.Type)).
        ToString() + " \n");

    Debug.WriteLine("e.VendorId = " + e.VendorId + " \n");
    Debug.WriteLine("e.ProductId = " + e.ProductId + " \n");

    switch (e.DeviceStatus) {
        case DeviceConnectionStatus.Connected:
            switch (e.Type) {
                case BaseDevice.DeviceType.Keyboard:
                    var keyboard = new Keyboard(e.Id, e.InterfaceIndex);
                    keyboard.KeyUp += Keyboard_KeyUp;
                    keyboard.KeyDown += Keyboard_KeyDown;
                    break;

                case BaseDevice.DeviceType.Mouse:
                    var mouse = new Mouse(e.Id, e.InterfaceIndex);
                    mouse.ButtonChanged += Mouse_ButtonChanged;
                    mouse.CursorMoved += Mouse_CursorMoved;
                    break;

                case BaseDevice.DeviceType.Joystick:
                    var joystick = new Joystick(e.Id, e.InterfaceIndex);
                    joystick.CursorMoved += Joystick_CursorMoved;
                    joystick.HatSwitchPressed += Joystick_HatSwitchPressed;
                    joystick.ButtonChanged += Joystick_ButtonChanged;
                    break;

                case BaseDevice.DeviceType.MassStorage:
                // see File System section
                    break;

                default:
                    break;
            }
            break;
```

```
        case DeviceConnectionStatus.Disconnected:
            Debug.WriteLine("Device Disconnected");
            //Unmount File System if it was mounted.
            break;

        case DeviceConnectionStatus.Bad:
            Debug.WriteLine("Bad Device");
            break;
    }
}
void Keyboard_KeyDown(Keyboard sender, Keyboard.KeyboardEventArgs args) {
    Debug.WriteLine("Key pressed: " + ((object)args.Which).ToString());
    Debug.WriteLine("Key pressed ASCII: " +
        ((object)args.ASCII).ToString());
}
void Keyboard_KeyUp(Keyboard sender, Keyboard.KeyboardEventArgs args) {
    Debug.WriteLine("Key released: " + ((object)args.Which).ToString());
    Debug.WriteLine("Key released ASCII: " + ((object)args.ASCII).ToString());
}
void Mouse_CursorMoved(Mouse sender, Mouse.CursorMovedEventArgs e) {
    Debug.WriteLine("Mouse moved to: " + e.NewPosition.X + ", " + e.NewPosition.Y);
}
    void Mouse_ButtonChanged(Mouse sender, Mouse.ButtonChangedEventArgs args) {
    Debug.WriteLine("Mouse button changed: " + ((object)args.Which).ToString());
}
    void Joystick_ButtonChanged(Joystick sender, Joystick.ButtonChangedEventArgs e) {
    Debug.WriteLine("Joystick button changed  = " + ((object)(e.Which)).ToString());
}
void Joystick_HatSwitchPressed(Joystick sender, Joystick.HatSwitchPressedEventArgs e) {
    Debug.WriteLine("Joystick direction  = " + ((object)(e.Direction)).ToString());
}
void Joystick_CursorMoved(Joystick sender, Joystick.CursorMovedEventArgs e) {
Debug.WriteLine("Joystick.move  = " + e.NewPosition.X + ", " + e.NewPosition.Y);
}
```

SECURING IOT

Having a secure IoT device is extremely important. There are all kinds of horror stories of bad things happening when devices are hacked into. How about someone controlling your car? Or getting into your security cameras? Sometimes attacks come from internal devices in your network that end up being a backdoor to your large secure network. When SITCore was designed, the security concerns were on the top of the list, even the name is Secure IoT Core… secure came before IoT. Security is not about just securing the network. It is also about the device itself.

SECURE STORAGE

There are 2 areas to store data internally and securely in SITCore's internal memory. The first one is the Secure Configuration. This is 128Kbytes on FEZ Bit organized as 4096 blocks of 32bytes. The user must write the data in blocks of 32bytes. A block can only be written once if that block was blank. It is possible to erase the entire config area, and then a rewrite is possible. Erase all will also wipe out this memory region.

SecureStorage library is found in GHIElectronics.TinyCLR.Devices.SecureStorage NuGet/namespace.

```
using System.Diagnostics;
using System.Text;
using GHIElectronics.TinyCLR.Devices.SecureStorage;

var configStorage = new SecureStorageController(SecureStorage.Configuration);
var dataBlock = new byte[configStorage.BlockSize];
dataBlock[0] = (byte)'F';
dataBlock[1] = (byte)'E';
dataBlock[2] = (byte)'Z';
if (configStorage.IsBlank(0)) {
    configStorage.Write(0, dataBlock);
}
else {
    Debug.WriteLine("Not blank!");
}
configStorage.Read(0, dataBlock);
var s = Encoding.UTF8.GetString(dataBlock, 0, 3);

Debug.WriteLine("Found: " + s);
```

There is also an OTP region, that is One Time Programmable. This area can never be erased, not even by erase all! This area is 2048 blocks of 32bytes each on FEZ Bit. The API is the same, just get the top controller. Here is the exact same code that writes FEZ to the first block, but remember once you run this code FEZ will be there forever!

```
using System.Diagnostics;
using System.Text;
using GHIElectronics.TinyCLR.Devices.SecureStorage;

var configStorage = new SecureStorageController(SecureStorage.Otp);
var dataBlock = new byte[configStorage.BlockSize];
dataBlock[0] = (byte)'F';
dataBlock[1] = (byte)'E';
dataBlock[2] = (byte)'Z';
if (configStorage.IsBlank(0)) {
    configStorage.Write(0, dataBlock);
}
else {
    Debug.WriteLine("Not blank!");
}
configStorage.Read(0, dataBlock);
var s = Encoding.UTF8.GetString(dataBlock, 0, 3);

Debug.WriteLine("Found: " + s);
```

IP PROTECTION

Intellectual Property protection is there to keep your application secure and away from bad use. The debug interface can be completely disabled so no one can load an application on your device, and it also prevents anyone from reading your device. Do NOT run this code as it will disable access to your device. This is reversable by going into the boot loader mode and “erase all”. This was used and explained in the USB Client chapter.

```
GHIElectronics.TinyCLR.Native.DeviceInformation.SetDebugInterface(DebugInterface.Disable);
```

There is also secure assemblies feature that comes in handy when external flash is used. There is no external flash in FEZ Bit and so all assemblies are automatically secure.

DATA SECURITY

Keeping data safe is done by supporting multiple ways of encrypting data. There is a section in this guide that covers Cryptography.

TLS is also supported internally and works seamlessly with the networking support. Note here that most IoT devices on the market use “secure TLS WiFi” modules that handle all the security, but the data is transferred between WiFi and the main processor unencrypted. This is not exactly secure! TinyCLR does all the TLS/security internally and any data going between WiFi and SITCore is transferred encrypted. More on TLS under the networking section.

On devices with external RAM, all data processing is happening inside the internal secure memory. External RAM is only used for graphics buffers.

THINKING SMALL

Many TinyCLR developers come from the PC/mobile world. They are used to writing code that runs fine on a PC, but then it will not run efficiently on an embedded device. The PC can be 4GHz with 16GB of RAM. A TinyCLR device can have less than 1% of the resources available on a PC.

MEMORY UTILIZATION

With limited RAM, developers should only use what they really need. PC programmers tend to make large buffers to handle the smallest tasks. Embedded Developers study what they need and only allocate the needed memory. For example, while reading data from a serial bus, we can very well use a 100 byte buffer to read the data or we can use a 1000 byte buffer. Both will work, but do we really need 1000 bytes? The answer depends on the system and what it is doing. The developer needs to put a bit more effort in determining how large of a buffer is needed to reserve for the incoming data.

Many of the TinyCLR drivers do a lot of buffering internally. For example, file system, UART, USB drivers all have internal buffers in native code, to keep the data ready until the developer uses the data from managed code, that is the C# side. To play a 5 megabyte MP3 file, we only need a 100 byte buffer that will read chunks from the file and pass to the MP3 decoder. Similar practices are done on the PC, except on the PC a working buffer can be 256K of RAM and on a small device it would be 100 bytes, or whatever is a reasonable minimum.

The good news here is that there is no memory hungry operating system that needs megabytes of free RAM to function. TinyCLR needs very little memory and so most of the system's memory is free for you to utilize.

OBJECT ALLOCATION

Allocating and freeing objects is very costly. This is true even on a PC, but on a 4Ghz machine it may not matter much. Only allocate objects when you really need them. Also, you are making an embedded device; therefore, a lot of objects that you will be using are always used. For example, if your system has an SPI display, always keep those objects alive and reuse them.

Consider this: There is a display that works over SPI bus and the display set pixel command is 3 bytes, which are 0x12 followed by x position followed by y position. The code will look like this:

```
void SetPixel(byte x, byte y) {
    spi.Write(new byte[] { 0x12, x, y });
}
```

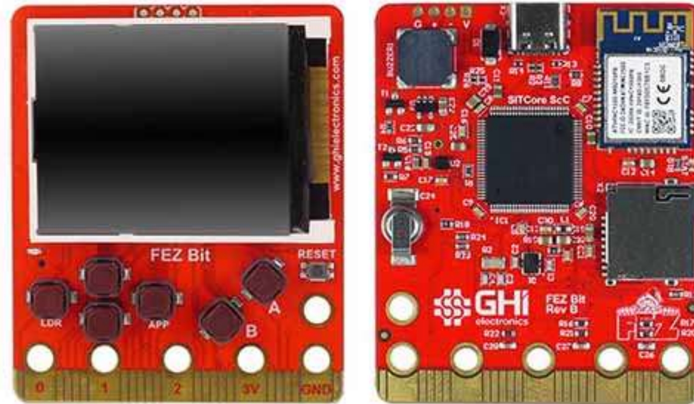
We have created a buffer on-the-fly to contain the three bytes and send to display. There is nothing wrong with that; but if this SetPixel function is called thousands (or tens of thousands) of times, then the system will create and dispose those thousands of buffers. Assuming we know this SetPixel method will be called thousands of times, we would instead keep a 3-byte-buffer alive and reuse it over and over.

```
var b3 = new byte[3];
void WriteReg(byte x, byte y) {
    b3[0] = 0x12;
    b3[1] = x;
    b3[2] = y;
    spi.Write(b3);
}
```

This code above will have zero impact on the garbage collector. Yes, we have indefinitely reserved 3 bytes, but that has no impact on the system.

FEZ BIT REFERENCE

Pin mapping for FEZ Bit is available through the Pins NuGet package. It includes the pins needed for Display, WiFi, and on-board LED.



Further technical docs: https://docs.ghielectronics.com/hardware/sitcore/sbc.html .

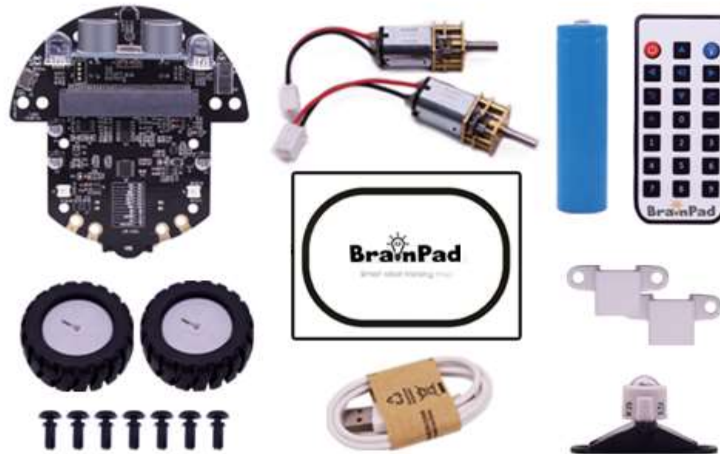
Schematics: https://docs.ghielectronics.com/hardware/sitcore/pdfs/fez-bit-rev-b-schematic.pdf .

BRAINBOT REFERENCE

BrainBot is a tiny robot with a rechargeable battery and several sensors.

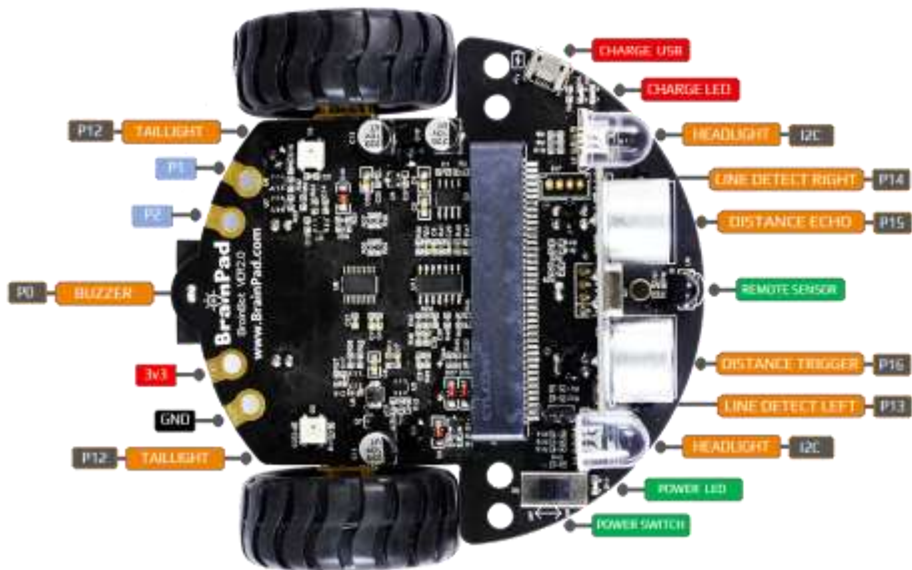
ASSEMBLY

The motors and wheels need to be mounted, using the included plastic rivets.



If you need details step-by-step, visit https://www.brainpad.com/lessons/brainbot-assembly/ .

PINOUT



BRAINCLIP REFERENCE

BrainClip includes sensors, cables, and a remote control.

All included sensors have GND to connect to GND and VCC to connect to 3V, which is the power source. A pad labeled NC means it is No Connect. Any other pads are for signals, such as OUT.

DIGITAL MODULES

- Button
- RGB LED (can be used with PWM)
- Motion Detector

ANALOG MODULES

- Rocker
- Light Sensor
- Sound Sensor

PWM MODULES

- RGB LED
- Buzzer

SPECIAL DIGITAL SIGNALS

- Ultrasonic Sensor
- LED Ring

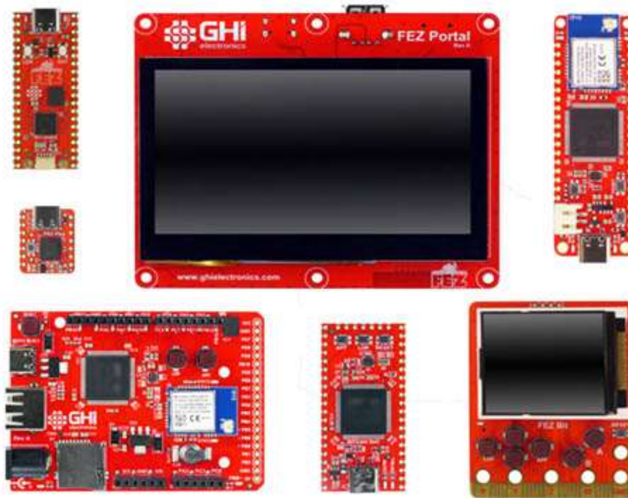
WHAT'S NEXT?

On the software front, you can continue to take advantage of your existing .NET C# experience to experiment further. You will quickly find yourself getting used to writing leaner and more optimized code.

The TinyCLR tutorials are your go-to going forward:
https://docs.ghielectronics.com/software/tinyclr/tutorials/intro.html .

As for hardware, there are hundreds of accessories you can add to your FEZ Bit. You can build all kinds of projects without soldering a single wire. Just search the web for "micro:bit accessory". The edge connector on FEZ Bit is micro:bit compatible. And once you are comfortable with circuits, you can get a breakout board and start wiring anything your heart desires!

Do not forget to add more FEZ boards to your collection. They range from the very-mall to full-featured.



https://www.ghielectronics.com/sitcore/sbc/

The forum is monitored by GHI Electronics' engineers and is full of very knowledgeable and friendly professionals. Jump in and show us what you are working on https://forums.ghielectronics.com/ .