Core Unit
Class Library Reference

Created April 23rd, 2019

Revision History

Date Revised	Revised Contents
2019/4/23	First release
2019/9/20	board Module information added
2020/2/19	Mu Editor is now the recommended programming environment Added module information to the Class Table appendix

Index

1. Getting Started

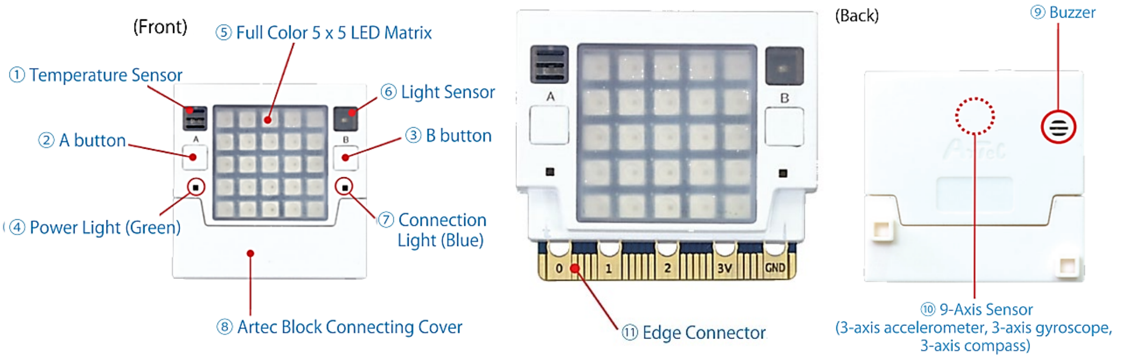
This manual is a reference guide for the ArtecRobo2.0 Studuino Class Library. Using this manual requires some prior understanding of the basics of Python.

2. Studuino:bit

The Studuino:bit is a computing device equipped with microcontrollers, sensors, LEDs, and other parts. MicroPython is one of the programming environments compatible with Studuino:bit.

2.1. Studuino:bit Layout

The layout of the Studuino:bit is shown below.


(Front)
⑤ Full Color 5 x 5 LED Matrix
① Temperature Sensor
⑥ Light Sensor
② A button
③ B button
④ Power Light (Green)
⑦ Connection Light (Blue)
⑧ Artec Block Connecting Cover
A
B
0
1
2
3V
GND
⑪ Edge Connector
(Back)
⑨ Buzzer
⑩ 9-Axis Sensor (3-axis accelerometer, 3-axis gyroscope, 3-axis compass)

The hardware parts operable with the Studuino:bit Class Library are ① Temperature Sensor, ② A button, ③ B button, ⑤ Full Color 5 x 5 LED Matrix, ⑥ Light Sensor, ⑨ Buzzer, ⑩ 9-Axis Sensor (3-axis accelerometer, 3-axis gyroscope, 3-axis compass), and ⑪ Edge Connectors.

From here on, the Full Color 5 x 5 LED Matrix will be referred to as the LED Display. The 9-Axis Sensor's three parts (3-axis accelerometer, 3-axis gyroscope, 3-axis compass) will be referred to as the Accelerometer, Gyroscope, and Geomagnetism Sensor respectively. The Edge Connectors will be called Ports.

2.2. MicroPython

MicroPython is version of Python developed specifically for use with microcontrollers. The programming syntax is identical to Python 3.0, but some Python libraries are not usable in MicroPython because they're not designed to work with the limited memory and CPU of a microcontrolller. However, some MicroPython-specific libraries (such as the library for GPIO microcontrollers) are included as standard.

MicroPython has been widely ported to different microcontrollers, and the Studuino:bit library utilizes a Studuino:bit specific version.

3. Development Environment

Some MicroPython development environments you can use with Studuino:bit are Mu Editor and uPyCraft. We provide an installer and instructions for using the Mu Editor on website at this URL:
https://www.artec-kk.co.jp/artecrobo2/en/software/python.php

4. Studuino:bit Class Library

The Studuino:bit class library is structured as follows.

Package	Module	Class	Hardware
pystubit	dsply	StuduinoBitDisplay	LED Display
	image	StuduinoBitImage	
	bzr	StuduinoBitBuzzer	Buzzer
	button	StuduinoBitButton	A/B Buttons
	sensor	StuduinoBitLightSensor	Light Sensors
		StuduinoBitTemperature	Temperature Sensor
		StuduinoBitAccelerometer	Accelerometer
		StuduinoBitGyro	Gyroscope
		StuduinoBitCompass	Geomagnetism Sensor (Compass)
	terminal	StuduinoBitTerminal	Port

Importing the board Module will let you use objects from the classes above (see section 5.1).

4.1. The StuduinoBitDisplay Class

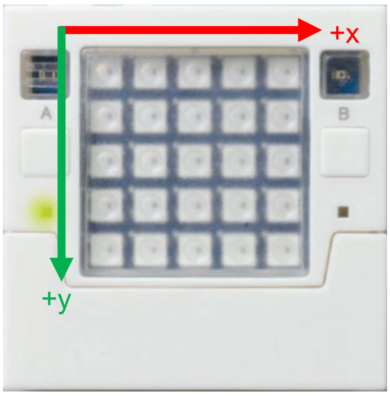
The StuduinoBitDisplay class is used to operate the 5 x 5 full color LED display on the front of the Studuino:bit Core Unit. This class can be used to make the LEDs display images, animations, and text.

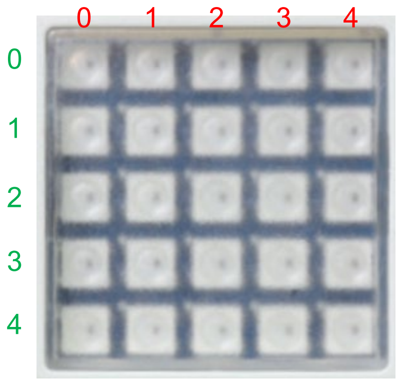
4.1.1. Constructors

Use this to make an object that operates the LED display.

```
from pystubit.dsply import StuduinoBitDisplay
display = StuduinoBitDisplay()
```

4.1.2. LED Settings





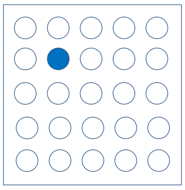
Use the method set_pixel(x, y, color) to change specific LEDs' settings. Each LED in the display is designated by a specific set of x-y coordinates (see image above). The x parameter in the set_pixel method picks the x-coordinate, y picks the y-coordinate, and color sets the color. The color parameter can be specified using a list, a tuple, or an integer. If using a tuple or list, use (R, G, B) or [R, G, B] to set the red, green and blue light levels. If using an integer, set the colors using a hex code as shown below. You can also use the clear() method to set the brightness of all LEDs in the display to 0 (off).

0xffffff

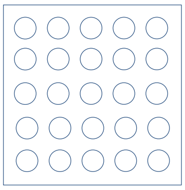
Red Green Blue

```
from pystubit.dsply import StuduinoBitDisplay
display = StuduinoBitDisplay()
display.set_pixel(1, 1, (0, 0, 10))
display.clear()
```

Running each of these methods should make the LED display respond as shown below.



Running the set_pixel method



Running the clear method

4.1.3. Retrieving Data from LEDs

Use the method get_pixel(x, y) to get information from specific LEDs. Set the x and y parameters to match the LED's x-y coordinates on the display to find out that LED's color in tuple format.

```
from pystubit.dsply import StuduinoBitDisplay
display = StuduinoBitDisplay()
display.set_pixel(1, 1, (0, 0, 10))
c1 = display.get_pixel(0, 0)
c2 = display.get_pixel(1, 1)
```

Variable c1 gets the data (0, 0, 0) from the LED at coordinates (0, 0) on the display, while variable c2 will get the data (0, 0, 10) from coordinates (1, 1).

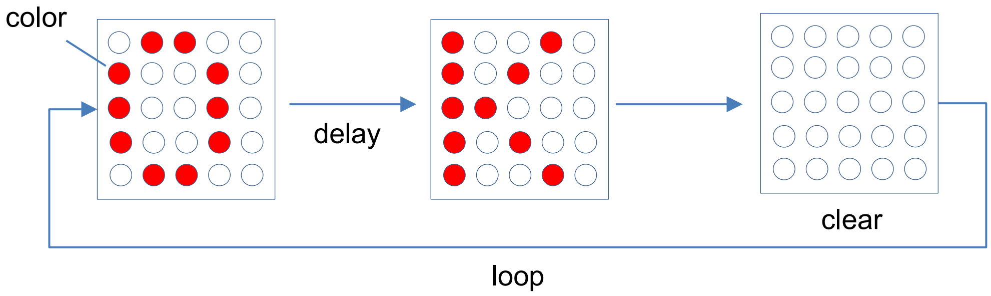
4.1.4. Animation

The method show(value, delay, wait=True, loop=False, clear=False, color=None) can be used to display letters and images with the LEDs.

Any string of numbers or letters you set in the value parameter will by displayed in sequence by the LEDs. If you instead set arrays for image objects (see section 4.2) in the value parameter, the LEDs will display each image in sequence. The delay parameter can be use to set the speed at which the display switches between images in milliseconds. If the wait parameter is set to True, other methods will be blocked until the display animation has finished. If it is set to False, other methods will run in the background. If the loop parameter is set to True, the animation will play on loop. If the clear parameter is set to True, the LED display will be cleared once the animation has finished. Set the color of the displayed image using the color parameter. The color parameter can be specified using a list, a tuple, or an integer.

```
from pystubit.dsply import StuduinoBitDisplay
display = StuduinoBitDisplay()
display.show('OK', 100, wait=True, loop=True, clear=True, color=(10, 0, 0))
```

Running the show method like this will make the LEDs display the letter O in red for 100 milliseconds, then the letter K for 100 milliseconds, and finally clear the display. This animation will play on loop.

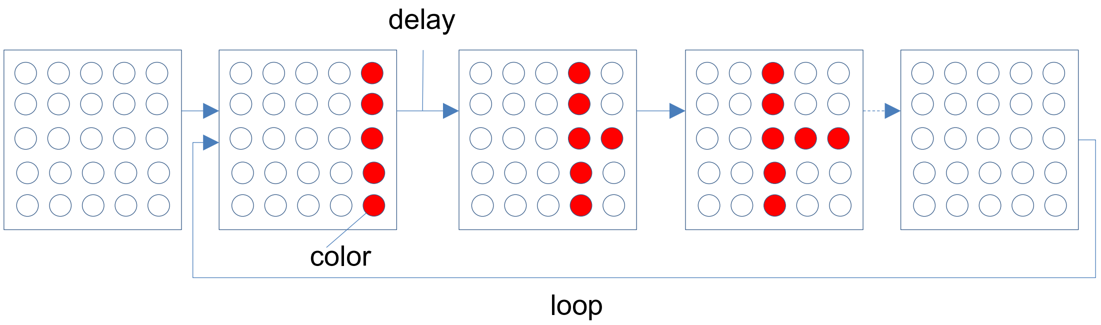


The method display.scroll(string,delay,wait=True,loop=False,color=None) can be used to make a series of characters scroll across the display.

You can set the string of letters and numbers to display in your scroll using the string parameter. Use the delay parameter to set how fast your string scrolls. If the wait parameter is set to True, other methods will be blocked until the display animation has finished. If it is set to False, other methods will run in the background. If the loop parameter is set to True, the animation will play on loop. Set the color of the displayed image using the color parameter. The color parameter can be specified using a list, a tuple, or an integer.

```
from pystubit.dsply import StuduinoBitDisplay
display = StuduinoBitDisplay()
display.scroll('Hello', 100, wait=True, loop=True, color=(10, 0, 0))
```

Running the scroll method like this will make the string Hello repeatedly scroll across the display in red, as shown below.

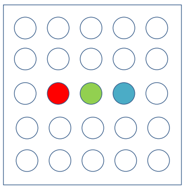


4.1.5. Display Power Settings

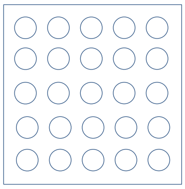
If the brightness of the all the LEDs in the display is set to 0, power to the display will be turned off. You can also use the off() method to switch the LED display's power OFF. Likewise, the on() method can be used to switch the LED display's power ON. You can check the status of the display's power using the is_on() method.

```
from pystubit.dsply import StuduinoBitDisplay
display = StuduinoBitDisplay()
display.set_pixel(1, 2, (10, 0, 0))
display.set_pixel(2, 2, (0, 10, 0))
display.set_pixel(3, 2, (0, 0, 10))
display.off()
ds1 = display.is_on()
display.on()
ds2 = display.is_on()
```

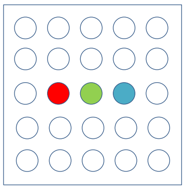
In the example below, the LED display has the LED at coordinates (1, 2) lit up in red, (2, 2) in green, and (3, 2) in blue. Running the off method turns off the LEDs, and running the on method turns them on again.



Before running the OFF method



Running the OFF method

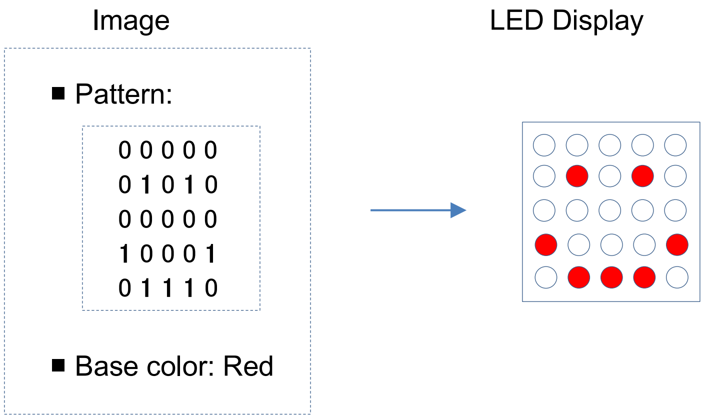


Running the ON method

The variable ds1 will be set to False, while ds2 will be set to True.

4.2. The StuduinoBitImage Class

Using the StuduinoBitImage class makes it easy to create images to display with your LEDs. Images are made up of a pattern and color data for the LEDs to display. Patterns are written using series of 0s (meaning OFF) and 1s (meaning ON). See the picture below for an example of how an image translate to the LED display.



Image

LED Display

■ Pattern:

```
0 0 0 0 0
0 1 0 1 0
0 0 0 0 0
1 0 0 0 1
0 1 1 1 0
```

■ Base color: Red

4.2.1. Constructors

There are four ways to make an image.

StuduinoBitImage(*,color)	Makes image using all 5 x 5 spaces (all 0).
StuduinoBitImage(string,*,color)	Makes a patter from a string of 0s and 1s in the string parameter.
StuduinoBitImage(width, height,*, color)	Makes an image using spaces within the specified width (the width parameter) and height (the height parameter).
StuduinoBitImage(width, height, buffer,*,color)	Makes an image using spaces within the specified width (the width parameter) and height (the height parameter) using a specified pattern (the buffer parameter).

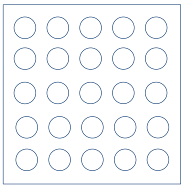
The color parameter can be used to set the base color in all constructors. The color parameter can be specified using a list, a tuple, or an integer. If the color variable has not been set, the base color will be red (31,0,0).

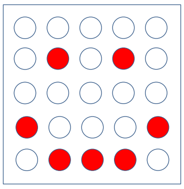
```
from pystubit.image import StuduinoBitImage
img1 = StuduinoBitImage()
img2 = StuduinoBitImage(’00000:’
                         ’01010:’
                         ’00000:’
                         ’10001:’
                         ’01110:’)
img3 = StuduinoBitImage(2,2)
img4 = StuduinoBitImage(2,2, bytearray([1,1,1,1]))
```

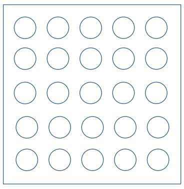
Images you’ve created can be displayed using display objects, as shown below.

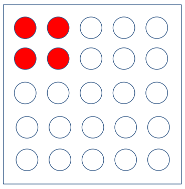
```
from pystubit.dsply import StuduinoBitDisplay
display.show(img1)
```

Variables img1, img2, img3, and img4 will appear on the LED display as shown below.


img1


img2


img3


img4

Constructors set with the string parameter can be displayed in a single line, as shown below.

```
img2 = StuduinoBitImage(’00000:01010:00000:10001:01110:’)
```

4.2.2. Finding Image Size

Find the size of an image using methids width() and height().

```
from pystubit.image import StuduinoBitImage
img = StuduinoBitImage(2,3)
w = img.width()
h = img.height()
```

The variable w will be set 2, while h will be set to 3.

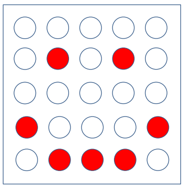
4.2.3. Image Settings

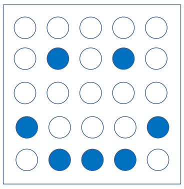
Use the set_pixel(x, y, value) method to edit an image pattern. Specify pixel positions using x-y coordinates just like the LEDs in the display, then set the value parameter to either 0 or 1. You can also use the set_pixel_color(x, y, color) method to change a single pixel’s color setting. Specify the pixel’s coordinates (x and y parameters) abd set the color in the color parameter.

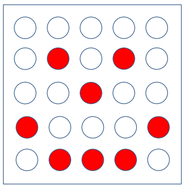
The set_base_color(color) method can be used to set the base color.

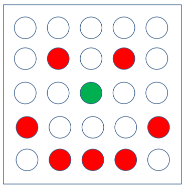
```
from pystubit.image import StuduinoBitImage
original = StuduinoBitImage('00000:01010:00000:10001:01110:')
img1.set_base_color((0,0,10))
img2.set_pixel(2,2,1)
img3.set_pixel_color(2,2,(0,10,0))
```

Variables original, img1, img2, and img3 will appear on the LED display as shown below. In img1 you can see that the base color has been changed to blue (0, 0, 10) with set_base_color. In img2 you can see that the pixel at (2, 2) has been set to 1 with set_pixel (making it display the base color). In img3 you can see that the pixel at (2, 2) has been changed to green (0, 10, 0) with set_pixel.


original


img1


img2


img3

4.2.4. Finding Image Data

Use the get_pixel(x, y) method to find pattern information for an image. Specifying the pixel's coordinates (x and y parameter) will return the pattern data for that pixel (0 or 1).
Use the get_pixel_color(x, y) method to find data for a single pixel in an image. Specify the pixel's coordinates (x and y parameters) to find its color in tuple format.

```
from pystubit.image import StuduinoBitImage
original = StuduinoBitImage('00000:01010:00000:10001:01110:')
v1 = img.get_pixel(0, 0)
c1 = img.get_pixel_color(0, 0)
v2 = img.get_pixel(1, 1)
c2 = img.get_pixel_color(1, 1)
```

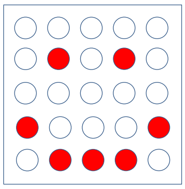
Variables v1 and c1 get the pattern data (0) and color data (0, 0, 0) from the LED at coordinates (0, 0) on the image, while variables v2 and c2 will get the pattern data (1) and color data (31, 0, 0) from coordinates (1, 1).

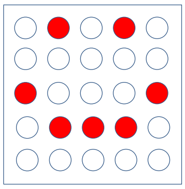
4.2.5. Manipulating Images

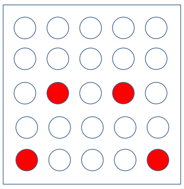
The methods shift_up(n), shift_down(n), shift_left(n), and shift_right(n) can be used to shift an image up, down, left, or right by the amount specified in the n parameter.

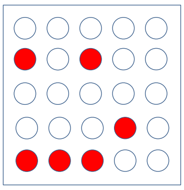
```
from pystubit.dsply import StuduinoBitImage
original = StuduinoBitImage(’00000:01010:00000:10001:01110:’)
up = original.shift_up(1)
down = original.shift_down(1)
left = original.shift_left(1)
right = original.shift_right(1)
```

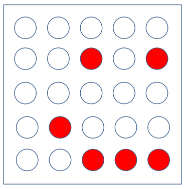
Variables original, up, down, left, and right will appear on the LED display as shown below. You can see that each image has been moved one line up, down, left, or right.


original


up


down

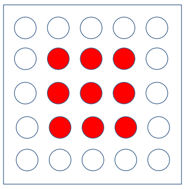

left

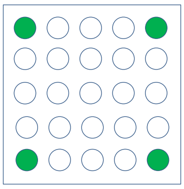

right

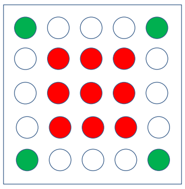
Using a + operator lets you combine several images into one.

```
from pystubit.dsply import StuduinoBitImage
img1 = StuduinoBitImage(’00000:01110:01110:01110:00000’,color=(10,0,0))
img2 = StuduinoBitImage(’10001:00000:00000:00000:10001’,color=(0,10,0))
img3 = img 1 + img2
```

Variables img1, img2, and img3 will appear on the LED display as shown below. You can see how img1 and img2 have been combined to make img3.


img1


img2


img3

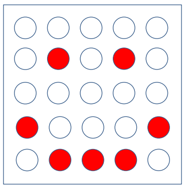
You can use the copy() method to make a copy of an image.

4.2.6. Included Images

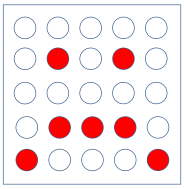
StuduinoBitImage lets you use a variety of patterns that are included in your Studuino:bit by default.

```
from pystubit.dsply import StuduinoBitImage
img1 = StuduinoBitImage.HAPPY
img2 = StuduinoBitImage.SAD
img3 = StuduinoBitImage.ANGRY
```

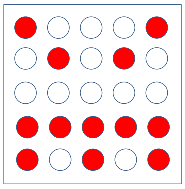
Variables img1, img2, and img3 will appear on the LED display as shown below.



img1



img2



img3

For a list of all the included images accessible with StuduinoBitImage, see section 5.3.

4.3. The StuduinoBitBuzzer Class

The StuduinoBitBuzzer class is used to control the buzzer.

4.3.1. Constructors

Use this to make an object that operates the Buzzer.

```
from pystubit.bzr import StuduinoBitBuzzer
buzzer = StuduinoBitBuzzer()
```

4.3.2. Adjusting the Sound

You can use the method on(sound, *, duration=None) to play sound from the buzzer. Use the sound parameter to set the frequency of the sound using a MIDI note number or note name. For a list of usable note numbers and names, see section 5.2. The duration parameter is used to set the length of the sound output in milliseconds. If the duration parameter is left unspecified, the off() method can be used to stop the buzzer.

```
from pystubit.bzr import StuduinoBitBuzzer
buzzer = StuduinoBitBuzzer()
buzzer.on('C4', duration=1000)
```

This will make the buzzer play the note C4 for 1 second.

4.3.3. Releasing PWMs

The Studuino:bit uses PWM for its sound output. It can use up to four PWMs simultaneously. If no PWM is available, you will be unable to use any buzzer objects you make. You can resolve this by using the release() method to release a PWM assigned to a different buzzer object.

4.4. The StuduinoBitButton Class

The StuduinoBitButton class is used to operate the two buttons on the face of the Studuino:bit's Main Unit.

4.4.1. Constructors

Use this to make an object that operates the buttons. Set A or B in the parameter.

```
from pystubit.button import StuduinoBitButton
button_a = StuduinoBitButton(’A’)
```

4.4.2. Finding Button States

Use the get_value() method to find the value (0/1) of a button as an integer, Using the is_pressed() method will return a True/False to tell you whether a button is being pressed. was_pressed() will return a True/False telling you whether a button was pressed in the past. The get_presses() method find how many times a button was pressed in the past.

```
from pystubit.button import StuduinoBitButton
button_a = StuduinoBitButton(’A’)
button_b = StuduinoBitButton(’B’)
print(’Press A button’)
while not button_a.is_pressed:
    pass
print(’A button is pressed’)
```

This will display the words “A button is pressed” on the standard display when the A Button has been pressed.

4.5. The StuduinoBitLightSensor Class

The StuduinoBitLightSensor class is used to operate the Light Sensor in the Main Unit.

4.5.1. Constructors

Use this to make an object that operates the Light Sensor.

```
from pystubit.sensor import StuduinoBitLightSensor
light_sensor = StuduinoBitLightSensor()
```

4.5.2. Finding Brightness

Use the method get_value() to find the analog value of the Light Sensor. It will be formatted as an integer from 0 to 4095.

```
from pystubit.sensor import StuduinoBitLightSensor
import time
light_sensor = StuduinoBitLightSensor()
while True:
    print(light_sensor.get_value())
    time.sleep_ms(500)
```

This will display the value of the Light Sensor on the standard display every 500 milliseconds.

4.6. The StuduinoBitTemperature Class

The StuduinoBitTemperature class is used to operate the Temperature Sensor in the Main

Unit.

4.6.1. Constructors

Use this to make an object that operates the Temperature Sensor.

```
from pystubit.sensor import StuduinoBitTemperature
temperature = StuduinoBitTemperature ()
```

4.6.2. Finding the Temperature

Use the method get_value() to find the analog value of the Temperature Sensor. It will be formatted as an integer from 0 to 4095. Use the method get_celsius(ndigits= 2) to get the temperature in degrees Celsius from the Temperature Sensor. The ndigits parameter can be used to set the number of decimal places.

```
from pystubit.sensor import StuduinoBitTemperature
import time
temperature = StuduinoBitTemperature ()
while True:
    print(temperature.get_value())
    print(temperature.get_celsius())
    time.sleep_ms(500)
```

This will display the analog value and the temperature in Celsius from the Temperature Sensor on the standard display every 500 milliseconds.

4.7. The StuduinoBitAccelerometer Class

The StuduinoBitAccelerometer class is used to operate the Accelerometer in the Main Unit.

4.7.1. Constructors

Use this to make an object that operates the Accelerometer.

```
from pystubit.sensor import StuduinoBitAccelerometer
acc = StuduinoBitAccelerometer(fs=’2g’, sf=’ms2’)
```

You can use the fs (full scale, meaning the maximum end of the measurement scale) and sf parameters in StuduinoBitAccelerometer constructors to set your units. Set 2g, 4g, 8g, or 16g in the fs parameter to measure up to 2G, 4G, 8G, or 16G of acceleration. It will be set to 2G by default. Set either ms2 or mg in the sf parameter to measure acceleration in either m/sec^2 or milli-Gs. It will be set to m/sec^2 by default.

4.7.2. Accelerometer Settings

Use the set_fs(value) method to set your Gyroscope’s maximum measurement. You can set the value parameter to 2g, 4g, 8g, or 16g. You can also use the set_sf(value) method to set the units of measurement. In can be set to either ms2 or mg.

4.7.3. Finding the Acceleration

Use the get_values(ndigits=2) method to find your Accelerometer’s values. The values will be returned in a tuple (x, y, z) format. The get_x(ndigits=2), get_y(ndigits=2) and get_z(ndigits=2) methods can each be used find the x-axis, y-axis and z-axis acceleration separately. Regardless of method, the ndigits parameter can be used to set the number of decimal places.

```
from pystubit.sensor import StuduinoBitAccelerometer
import time
acc = StuduinoBitAccelerometer()
while True:
      print(acc.get_values())
      time.sleep_ms(500)
```

This will display the Accelerometer values on the standard display every 500 milliseconds.

4.8. The StuduinoBitGyro Class

The StuduinoBitGyro class is used to operate the Gyroscope in the Main Unit.

4.8.1. Constructors

Use this to make an object that operates the Gyroscope.

```
from pystubit.sensor import StuduinoBitGyro
gyro = StuduinoBitGyro (fs=’250dps’, sf=’dps’))
```

You can use the fs (maximum end of the measurement scale) and sf parameters in StuduinoBitGyro constructors to set your units. Set 250dps, 500dps, 1000dps, or 2000dps in the fs parameter to measure up to 250 deg/s, 500 deg/s, 1000 deg/s, or 2000 deg/s of angular velocity. It will be set to 250 deg/s by default. Set either dps or rps in the sf parameter to take measurements in either deg/s or rad/s. It will be set to deg/s by default.

4.8.2. Gyroscope Settings

Use the set_fs(value) method to set your Gyroscope’s maximum measurement. You can set the value parameter to 250dps, 500dps, 1000dps, or 2000dps. You can also use the set_sf(value) method to set the units of measurement. In can be set to either dps or rps.

4.8.3. Finding Angular Velocity

Use the method get_values(ndigits=2) to find the value of the Gyroscope. The values will be returned in the units specificed by either the constructor or the set_sf method, in a tuple (x, y, z) format.

The get_x(ndigits=2), get_y(ndigits=2) and get_z(ndigits=2) methods can each be used find the x-axis, y-axis and z-axis angular velocity separately. Regardless of method, the ndigits parameter can be used to set the number of decimal places.

```
from pystubit.sensor import StuduinoBitGyro
import time
gyro = StuduinoBitGyro ()
while True:
      print(gyro.get_values())
      time.sleep_ms(500)
```

This will display the Gyroscope's values on the standard display every 500 milliseconds.

4.9. The StuduinoBitCompass Class

The StuduinoBitCompass class is used to operate the geomagnetism sensor (or Compass) in the Main Unit.

4.9.1. Constructors

Use this to make an object that operates the Compass.

```
from pystubit.sensor import StuduinoBitCompass
compass = StuduinoBitCompass ()
```

4.9.2. Calibration

Calibrate the Compass using the calibrate() method. The calibration process will use the LED display.

Use the is_calibrated() method to find out whether the Compass has been calibrated. Use the clear_calibration() method to clear any existing calibration data.

4.9.3. Finding Directions

Use the heading() method to find directional information (in degrees 0-360). If your Compass has not been calibrated, calibration will begin automatically when you run the heading() method. 0° is defined as when the Main Unit is placed with the LED Display facing up and the USB connector facing due south. Angles increase from there in a clockwise fashion.

4.9.4. Finding Compass Values

Use the method get_values() to find the Compass's values. The values will be returned in a tuple (x, y, z) format, using μT (micro tesla) units.

The get_x(), get_y() and get_z() methods can each be used find what direction the x-axis, y-axis and z-axis are facing separately.

```
from pystubit.sensor import StuduinoBitCompass
import time
compass = StuduinoBitCompass ()
while True:
        print(compass.get_values())
        time.sleep_ms(500)
```

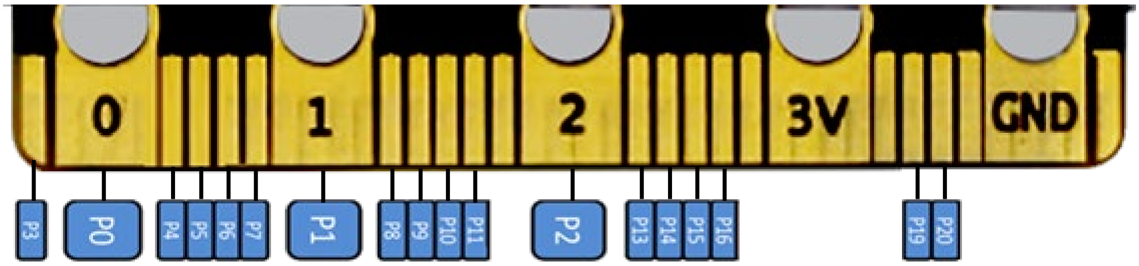
This will display the Compass's values on the standard display every 500 milliseconds.

4.10. The StuduinoBitTerminal Class

The StuduinoBitTerminal class is used to operate the ports in the Main Unit.

4.10.1. Constructors

StuduinoBitTerminal constructors can be used to make objects that operate all the ports on Main Unit from P0 to P20, except for P17 and P18. The ports operable with the StuduinoBitTerminal class are shown below, P0-P20.



Use this to make an object that operates a specified port between P0 and P20.

```
from pystubit.terminal import StuduinoBitTerminal
p0 = StuduinoBitTerminal( 'P0' )
```

The input/output capabilities of the ports between P0 and P20 are listed below.

Port	I/O Type	Port	I/O Type
P0	Digital Input/Output and Analog Input	P10	Digital Input/Output
P1	Digital Input/Output and Analog Input	P11	Digital Input/Output
P2	Digital and Analog Input	P12	Digital Input/Output
P3	Digital and Analog Input	P13	Digital Input/Output
P4	Digital Input/Output	P14	Digital Input/Output
P5	Digital Input/Output	P15	Digital Input/Output
P6	Digital Input/Output	P16	Digital Input/Output
P7	Digital Input/Output	P19	Digital Input/Output
P8	Digital Input/Output	P20	Digital Input/Output

P9	Digital Input/Output		

4.10.2. Digital Input/Output

If the port object you've made is a digital input/output type, you can use the write_digital(value) method to send digital signals with it. The value parameter can be set to either 0 or 1. You can also use the read_digital() method to read digital signals. Use the set_analog_hz() method to pick a PWM output frequency to use, and then use the write_analog(value) method to send PWM signals.

4.10.3. Digital Input/Output and Analog Input

If the port object you've made is a digital input/output and analog input type, you can use the digital input/output methods described above, as well as the read_analog() method to read analog signals.

4.10.4. Digital and Analog Input

If the port object you've made is a digital and analog input type, you can use only the read_digital() and read_analog() methods with it.

5. Appendices

5.1. The board Module

The board module contains the definitions of the objects used to operate the Studuino:bit hardware. Make a declaration as shown to use an object.

```
from pystubit.board import *
```

The objects defined by the board module are listed below.

Object	Hardware
display	LED Display
Image (Class)	LED Display (Image)
buzzer	Buzzer
button_a, button_b	A/B Buttons
lightsensor	Light Sensors
temperature	Temperature Sensor
accelerometer	Accelerometer
gyro	Gyroscope
compass	Geomagnetism Sensor (Compass)
P0-16, P19, P20	Edge Connector
pin0-pin3	Edge Connector for micro:bit (★)

★: The following objects cannot be used together at the same time: pin0 and p0, pin1 and p1, pin2 and p2, pin3 and p3.

★: The objects pin0-pin3 can use the same method as objects p0-p3, but the read_analog() method will return values from 0 to 1023.

5.2. Sound Output

The MIDI note numbers and note names used in the StuduinoBitBuzzer class are as follows.

MIDI	Note	MIDI	Note	MIDI	Note	MIDI	Note	MIDI	Note	MIDI	Note	MIDI	Note
48	C3	60	C4	72	C5	84	C6	96	C7	108	C8	120	C9
49	CS3	61	CS4	73	CS5	85	CS6	97	CS7	109	CS8	121	CS9
50	D3	62	D4	74	D5	86	D6	98	D7	110	D8	122	D9
51	DS3	63	DS4	75	DS5	87	DS6	99	DS7	111	DS8	123	DS9
52	E3	64	E4	76	E5	88	E6	100	E7	112	E8	124	E9
53	F3	65	F4	77	F5	89	F6	101	F7	113	F8	125	F9
54	FS3	66	FS4	78	FS5	90	FS6	102	FS7	114	FS8	126	FS9
55	G3	67	G4	79	G5	91	G6	103	G7	115	G8	127	G9
56	GS3	68	GS4	80	GS5	92	GS6	104	GS7	116	GS8		
57	A3	69	A4	81	A5	93	A6	105	A7	117	A8		
58	AS3	70	AS4	82	AS5	94	AS6	106	AS7	118	AS8		
59	B3	71	B4	83	B5	95	B6	107	B7	119	B8		

5.3. Included Images

The images included in the Studuino:bit library are as follows.

StuduinoBitImage.HEART

StuduinoBitImage.HEART_SMALL

StuduinoBitImage.HAPPY

StuduinoBitImage.SMILE

StuduinoBitImage.SAD

StuduinoBitImage.CONFUSED

StuduinoBitImage.ANGRY

StuduinoBitImage.ASLEEP

StuduinoBitImage.SURPRISED

StuduinoBitImage.SILLY

StuduinoBitImage.FABULOUS

StuduinoBitImage.MEH

StuduinoBitImage.YES

StuduinoBitImage.NO

StuduinoBitImage.CLOCK12,

StuduinoBitImage.CLOCK11,

```
StuduinoBitImage.CLOCK10,
StuduinoBitImage.CLOCK9,
StuduinoBitImage.CLOCK8,
StuduinoBitImage.CLOCK7,
StuduinoBitImage.CLOCK6,
StuduinoBitImage.CLOCK5,
StuduinoBitImage.CLOCK4,
StuduinoBitImage.CLOCK3,
StuduinoBitImage.CLOCK2,
StuduinoBitImage.CLOCK1
StuduinoBitImage.ARROW_N,
StuduinoBitImage.ARROW_NE,
StuduinoBitImage.ARROW_E,
StuduinoBitImage.ARROW_SE,
StuduinoBitImage.ARROW_S,
StuduinoBitImage.ARROW_SW,
StuduinoBitImage.ARROW_W,
StuduinoBitImage.ARROW_NW
StuduinoBitImage.TRIANGLE
StuduinoBitImage.TRIANGLE_LEFT
StuduinoBitImage.CHESSBOARD
StuduinoBitImage.DIAMOND
StuduinoBitImage.DIAMOND_SMALL
StuduinoBitImage.SQUARE
StuduinoBitImage.SQUARE_SMALL
StuduinoBitImage.RABBIT
StuduinoBitImage.COW
StuduinoBitImage.MUSIC_CROTCHET
StuduinoBitImage.MUSIC_QUAVER
StuduinoBitImage.MUSIC_QUAVERS
StuduinoBitImage.PITCHFORK
StuduinoBitImage.XMAS
StuduinoBitImage.PACMAN
StuduinoBitImage.TARGET
StuduinoBitImage.TSHIRT
```

StuduinoBitImage.ROLLERSKATE

StuduinoBitImage.DUCK

StuduinoBitImage.HOUSE

StuduinoBitImage.TORTOISE

StuduinoBitImage.BUTTERFLY

StuduinoBitImage.STICKFIGURE

StuduinoBitImage.GHOST

StuduinoBitImage.SWORD

StuduinoBitImage.GIRAFFE

StuduinoBitImage.SKULL

StuduinoBitImage.UMBRELLA

StuduinoBitImage.SNAKE

5.4. Class Table

This table lists the methods available for each class, and which objects from the board module the classes correspond to.

Function	Class	The board Module	Method	Instructions
Buttons	StuduinoBitButton	button_a button_b	_init_(ab)	Used to make objects that operate the A/B buttons. The ab parameter can be set to either A or B.
			get_value()	Returns a 0 if the the button is being pressed and 1 if not.
			is_pressed()	Returns a True if the button is being pressed.
			was_pressed()	Button objects store the information that the button has been pressed. This method returns a True if the button is has been pressed in the past. The information is reset when this method is called.
			get_presses()	Button objects store the information that the button has been pressed. This method returns the number of times a button has been pressed in the past. The count is reset when this method is called.
LED Display (Full color)	StuduinoBitDisplay	display	_init_()	Makes display objects.
			get_pixel(x, y)	Returns the color of the LED at row x of column y. The color will be displayed in (R,G,B).
			set_pixel(x, y, color)	Set the color of the LED at row x of column y. The parameter can be set using (R,G,B), [R,G,B] or #RGB.
			clear()	Set the brightness of all LEDs in the display to 0 (off).
			show(iterable, delay=400, *, wait=True, loop=False, clear=False, color=None)	Displays the iterable parameter (an image, string, or number) in sequence.
			scroll(string, delay=150, *, wait=True, loop=False, color=None)	Scrolls the value parameter (letters/numbers) across the display horizontally.
			on()	Turn on power to the LED display.
			off()	Turn off power to the LED display. (This allows you to used GPIO terminals

				connected to the display for other purposes.)
			is_on()	Returns True if the display’ s power is ON, and False if it’ s OFF.

Image	StuduinoBitImage	Image	_init_(string, color=None)
			init(width=None, height=None, buffer=None, , color=None)
				Makes image objects using a pattern written in 0s (LED OFF) and 1s (LED ON) in the string parameter.
				StuduinoBitImage('01100:10010:11110:10010:10010:', color=(0,0,10))
				Makes an image object using spaces within the specified width (the width parameter) and height (the hight parameter).
				StuduinoBitImage(2, 2,bytearray([0,1,0,1])
				StuduinoBitImage(3, 3)
				As with micro:bit, any number 0–9 can be used, but 1–9 all translate to ON, while 0 is OFF. If the color variable has not been set, (RGB)= (31,0,0).

			width()	Returns how wide the image is in columns.
			height()	Returns how tall the image is in rows.
			set_pixel(x, y, value)	The value parameter sets the value of the pixel at coordinates (x, y) in the image. The value parameter can be set to either 0 (OFF) or 1 (ON).
			set_pixel_color(x, y, color)	The color parameter sets the color of the pixel at coordinates (x, y) in the image. The color parameter can be set using (R, G, B), [R, G, B] or #RGB.
			get_pixel(x, y)	Returns the value of the pixel at coordinates (x, y) in the image.
			get_pixel_color(x, y, hex=False)	Returns the color of the pixel at coordinates (x, y) in the image. If the hex parameter is set to False, the color will be written in (R,G,B). If set to True, it will be written in #RGB.
			set_base_color(self, color)	The color parameter sets the color of all the pixels in the image. The color parameter can be set using (R, G, B), [R, G, B] or #RGB.
			shift_left(n)	Makes a next image by shifting the current image a number of spaces left set in the n parameter.
			shift_right(n)	Identical to shift_left(−n).
			shift_up(n)	Makes a next image by shifting the current image a number of spaces up set in the n parameter.
			shift_down(n)	Identical to shift_up(−n).
			copy()	Returns a complete copy of the image.
			repr(image)	Get a compact string representing the image.
			str(image)	Get a readable string representing the image.

			+	Combine all the pixels from two images to make a new image.
				Included Images (See 5.3)
Port	StuduinoBitTerminal	See port modules below.	_init_(pin)	Make a terminal object that applies to a designated Studuino:bit port P0–P16, P19, or P20.
Digital Input/Output and Analog Input （※）	StuduinoBitAnalogDigitalPin	p0, p1, p2, p3 Note: p2 and p3 cannot be used for digital input/output. Using the write_digital method with them will cause an error. The read_digital method is implemented with analog input, so it can be used.	write_digital(value)	Sets digital signals to High if the value parameter is 1, and Low if the value parameter is 0.
			read_digital()	Reads digital signals. Returns High if the received value is 0, and Low if it’ s 1.
			write_analog(value)	Sends PWM signals through the port. The value parameter can be set from 0 (0%) to 1023 (100%).
			set_analog_period(period, timer=–1)	Sets the period of the PWM signal in milliseconds.
			set_analog_period_microseconds(period, timer=–1)	Sets the period of the PWM signal in microseconds.
			set_analog_hz(hz, timer=–1)	Sets the period of the PWM signal by frequency.
			status()	Shows the current usage status of the PWMs.
			read_analog(mv=False)	Reads voltage from the port and, if the mv parameter is set to False, returns it as an integer between 0 (0V) and 4096 (3.3V). If mv=True, the voltage will be written in mV.
Digital Input/Output (★)	StuduinoBitDigitalPin	p4–p16, p19, p20	write_digital(value)	Sets digital signals to High if the value parameter is 1, and Low if the value parameter is 0.
			read_digital()	Reads digital signals. Returns High if the received value is 0, and Low if it’ s 1.
			write_analog(value)	Sends PWM signals through the port. The value parameter can be set from 0 (0%) to 1023 (100%).
			set_analog_period(period, timer=–1)	Sets the period of the PWM signal in milliseconds.
			set_analog_period_microseconds(period, timer=–1)	Sets the period of the PWM signal in microseconds.

			set_analog_hz(hz, timer=-1)	Sets the period of the PWM signal by frequency.
			status()	Shows the current usage status of the PWMs.
Buzzer	StuduinoBitBuzzer	buzzer	_init_()	Make objects that operate the Buzzer.

			on(sound, *, duration=-1)	Set the Buzzer’s sound in the sound and duration parameters. sound can be set using note names (C3-G9), MIDI note numbers (48-127), or frequency (as an integer), duration can be set in ms. If the duration parameter is left unspecified, the Buzzer will keep playing until you call the off method.
				bzr.on('50', duration=1000) # The buzzer will play the note corresponding to MIDI note number 50 for 1 second.
				bzr.on('C4', duration=1000) # The buzzer will play the note C4 for 1 second
				bzr.on(440)

			off()	Make the Buzzer stop playing sound.
Temperature Sensor	StuduinoBitTemperature	temperature	_init_()	Make objects that operate the Temperature Sensor.
			get_value()	Returns the value (0-4095) of the Temperature Sensor in the Main Unit.
			get_celsius()	Returns the value of the Temperature Sensor in the Main Unit in degrees Celsius.
Light Sensors	StuduinoBitLightSensor	lightsensor	_init_()	Make objects that operate the Light Sensor.
			get_value()	Returns the value (0-4095) of the Light Sensor in the Main Unit.
Accelerometers	StuduinoBitAccelerometer	accelerometer	_init_(fs=’2G’, sf=’mg2’)	Make an Accelerometer object by setting the fs parameter to 2g, 4g, 8g, or 16g to set maximum acceleration to measure, and the sf parameter to mg or ms2 to pick your units of measurement. The default settings are fs=2G and sf=ms2.
			get_x	Measure acceleration along the x-axis.
			get_y	Measure acceleration along the y-axis.
			get_z	Measure acceleration along the x-axis.
			get_values()	=(get_x(), get_y(), get_z())
			set_axis(mode)	mode can be set with one of the following strings: sbmp (for Studuino:bit MicroPython), sbs (for Studuino:bit Software) or mb (for micro:bit). It will be set to sbmp by default.
			set_fs(value)	Set the maximum acceleration to measure with 2g, 4g, 8g, or 16g.
			set_sf(value)	Set the units of measurement with mg or ms2.

Gyroscope	StuduinoBitGyro	gyro	_init_(fs=’250dps’, sf=’dps’)	Make a Gyroscope object by setting the fs parameter to 250dps, 500dps, 1000dps, or 2000dps to set maximum angular velocity to measure, and the sf parameter to dps or rps to pick your units of measurement. The default settings are fs=250dps and sf=dps.
			get_x	Measure angular velocity along the x–axis.
			get_y	Measure angular velocity along the y–axis.
			get_z	Measure angular velocity along the x–axis.
			get_values()	=(get_x(), get_y(), get_z())
			set_axis(mode)	mode can be set with one of the following strings: sbmp (for Studuino:bit MicroPython), sbs (for Studuino:bit Software) or mb (for micro:bit). It will be set to sbmp by default.
			set_fs(value)	Set the maximum angular velocity to measure with 250dps, 500dps, 1000dps, or 2000dps.
			set_sf(value)	Set the units of measurement with dps or rps.
Compass	StuduinoBitCompass	compass	_init_()	Make objects that operate the Compass. The maximum measurable value is ±4500μT, and the units are μT (micro teslas).
			get_x	Finds the magnetic force along the x–axis.
			get_y	Finds the magnetic force along the y–axis.
			get_z	Finds the magnetic force along the z–axis.
			get_values()	=(get_x(), get_y(), get_z())
			set_axis(mode)	mode can be set with one of the following strings: sbmp (for Studuino:bit MicroPython), sbs (for Studuino:bit Software) or mb (for micro:bit). It will be set to sbmp by default.
			calibrate()	Starts the Compass calibration process. Rotate the Main Unit until the LED Display draws a full circle around its border.
			is_calibrated()	Returns false True if the Compass has been calibrated and False if it hasn’t.
			clear_calibration()	Erases existing calibration data.
			heading()	Finds which direction the Compass is facing. In integers 0–360° counting clockwise, with 180° being North.
BLE	StuduinoBitBLE			T.B.D
Wireless	StuduinoBitRadio	There are no	_init()_	Make objects that operate the unit’s wireless communications.

Communications		objects for this class.	on()	Turns on wireless communications. Wireless communications must be explicitly called up as they consume power and take up memory that may be necessary for other functions.
			off()	Turns off wireless communications, conserving the unit’s power and memory.
			The following methods cannot be used unless wireless communications have been turned on.	
			start(group)	Set the group parameter to a number between 0 and 255 to pick a group and begin wireless communications.
			send_number(n)	Broadcasts a number. The number will be a 32-bit integer.
			send_value(s, n)	Broadcasts a number and variable name. The number will be a 32-bit integer, the the variable name can be up to 13 characters long.
			send_string(s)	Broadcasts a string. The string can be up to 19 characters long.
			send_buffer(buf)	Broadcasts a byte sequence. It can be up to 19 bytes.
			recv()	Receives data. If no data has been received, returns None.
			group(group=-1)	Set the group parameter to a number between 0 and 255 to pick a group.
Wi-Fi	StuduinoBitWiFi			T.B.D

(★) The individual Digital Input/Output, Analog Input, and Digital and Analog Input/Output classes are intended to be made into instances. Instances are intended up be made via the Terminal class.