Studuino:bit クラスライブラリ
リファレンス

2019/04/23　作成

2023/03/13　改訂

改訂履歴

改訂日付	改訂内容
2019/04/23	初版発行
2019/09/20	board モジュールの説明を追加
2020/02/19	プログラミング推奨環境を Mu エディタに変更 付録のクラス一覧にモジュール列を追加
2020/03/16	StuduinoBitRadio クラスの説明追加
2020/07/07	誤字修正
2023/02/08	お問い合わせ先の変更
2023/03/13	クラス一覧にある、read_digital()メソッドの誤植修正

目次

1. はじめに

本書は、ArtecRobo2.0 用基板 Studuino:bit を制御するクラスライブラリのリファレンスです。本書を読み進めるにあたり、Python の基本文法の理解が必要になります。

2. Studuino:bit

Studuino:bit は、マイコンとセンサーや LED 等を搭載したコンピュータ・デバイスで、使用できるプログラミング環境の一つに MicroPython があります。

2.1. Studuino:bit の構成

Studuino:bit の構成を下図に示します。

Studuino:bit クラスライブラリを使用して制御するハードウェアは、①温度センサー、②A ボタン、③B ボタン、⑤フルカラーLED マトリックス 5 x 5、⑥光センサー、⑨電子ブザー、⑩9 軸センサー（3 軸加速度＋3 軸ジャイロ＋3 軸コンパス）⑪エッジコネクターです。以下では簡単のため、「フルカラーLED マトリックス 5 x 5」を「LED ディスプレイ」、「9 軸センサー（3 軸加速度＋3 軸ジャイロ＋3 軸コンパス）」をセンサー毎に「加速度センサー」、「ジャイロセンサー」、「地磁気センサー」と記し、「エッジコネクター」を「端子」と記します。

2.2. MicroPython

MicroPython は、マイコンを簡単に制御するために開発されたマイコン制御用の Python です。プログラムの構文は Python3.0 と同様ですが、メモリや CPU の性能に制限があるマイコン用に実装されており、MicroPython では使用できない Python の標準ライブラリもあります。また一方で、マイコンの GPIO を制御するライブラリなど MicroPython 特有のライブラリが標準で組み込まれています。

MicroPython は、様々なマイコンボードに移植されており、Studuino:bit ライブラリは、Studuino:bit 用に移植した MicroPython で動作します。

3. 開発環境

Studuino:bit 上で動作する MicroPython の開発環境の一つとして、Mu エディタや uPyCraft があります。Mu エディタは、下記ページでインストーラと使用方法を提供しています。

https://www.artec-kk.co.jp/artecrobo2/ja/software/python.php

4. Studuino:bit クラスライブラリ

Studuino:bit クラスライブラリは、下記の構成になっています。

パッケージ	モジュール	クラス	ハードウェア
pystubit	dsply	StuduinoBitDisplay	LED ディスプレイ
	image	StuduinoBitImage	
	bzr	StuduinoBitBuzzer	電子ブザー
	button	StuduinoBitButton	ボタン A、ボタン B
	sensor	StuduinoBitLightSensor	光センサー
		StuduinoBitTemperature	温度センサー
		StuduinoBitAccelerometer	加速度センサー
		StuduinoBitGyro	ジャイロセンサー
		StuduinoBitCompass	地磁気センサー
	terminal	StuduinoBitTerminal	端子

また、board モジュールをインポートすることで上記クラスのオブジェクトを使用できます（5.1 参照）。

4.1. StuduinoBitDisplay クラス

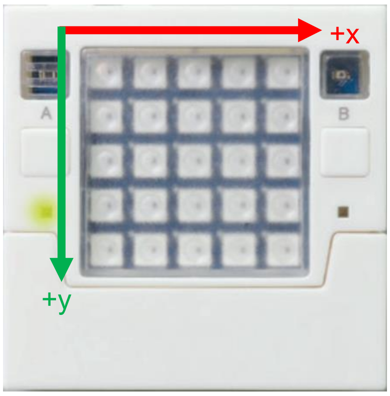
StuduinoBitDisplay クラスは、基板の前面にある 5×5 のフルカラーLED ディスプレイを制御するクラスです。また、本クラスを使用する事で、イメージ、アニメーション、テキストを LED ディスプレイに表示することができます。

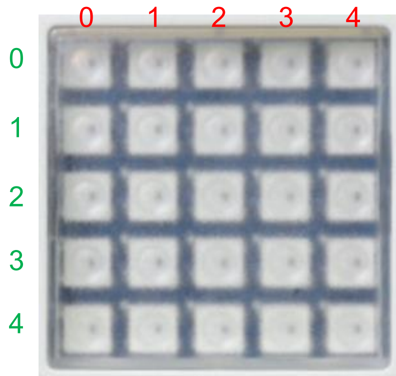
4.1.1. コンストラクタ

LED ディスプレイを制御するオブジェクトを作成します。

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

4.1.2. LED の設定

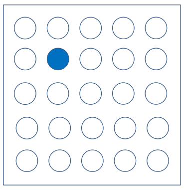




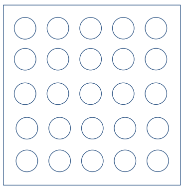
set_pixel(x, y, color)メソッドで、LED を設定します。各 LED の位置は x-y 座標系で決められています（上図）。set_pixel メソッドの x 引数に x 列、y 引数に y 行、color 引数に LED の色を設定します。color 引数は、タプルまたはリストまたは整数で指定します。タプルまたはリストで設定する場合は、R（赤）、G（緑）、B（青）の明るさをそれぞれ(R, G, B)または[R, G, B]で設定します。整数の場合は、16 進数表記で下図のように色を設定します。また、clear()メソッドで、すべての LED の明るさを 0（オフ）に設定することができます。


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

LED ディスプレイには、各メソッド実行時に下図のように表示されます。



set_pixel メソッド実行時



clear メソッド実行時

4.1.3. LED 情報の取得

get_pixel(x, y)メソッドで、LED の情報を取得します。LED ディスプレイ上の(x, y)の位置を x 引数と y 引数に設定し、LED の色をタプルで取得します。

```
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)
```

変数 c1 には、LED ディスプレイの位置(0, 0)の LED の情報(0, 0, 0)が、変数 c2 には位置(1,1)の LED の情報(0, 0, 10)が代入されます。

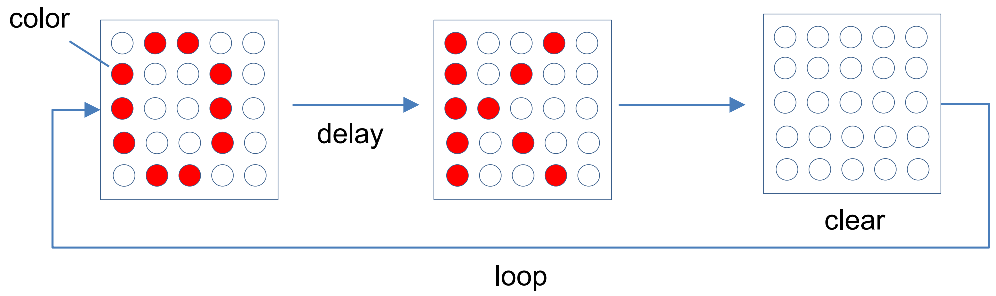
4.1.4. アニメーション

show(value, delay, wait=True, loop=False, clear=False, color=None)メソッドで、LED ディスプレイに文字やイメージを表示することができます。

value 引数に数字または文字列を設定した場合、設定した数字または文字が LED ディスプレイに切り替わり表示されます。また、value 引数に image オブジェクト（4.2 参照）の配列を設定した場合、各イメージが LED ディスプレイに切り替わり表示されます。delay 引数で表示が切り替わる速さをミリ秒単位で設定できます。wait 引数に True を設定した場合、表示が完了するまでメソッドはブロックします。False を設定した場合、バックグラウンドで実行されます。loop 引数に True を設定した場合、表示を繰り返します。clear 引数に True を設定した場合、表示終了後 LED ディスプレイがクリアされます。表示するイメージの色は、color 引数で設定します。color 引数は、タプルまたはリストまたは整数で設定できます。

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

show メソッドを実行すると下図のように、LED ディスプレイに赤色で文字'O'が表示され 100 ミリ秒後に文字'K'が表示され、100 ミリ秒後に表示がクリアされます。この表示をループします

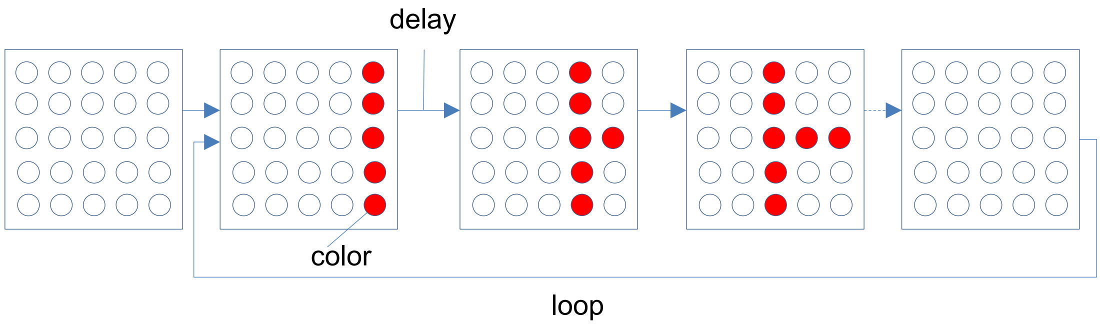


display.scroll(string,delay,wait=True,loop=False,color=None)メソッドで、LED ディスプレイに文字をスクロールで表示することができます。

string 引数に LED ディスプレイにスクロールさせたい英数文字列を設定します。delay 引数にスクロールの速さを設定します。wait 引数に True を設定した場合、表示が完了するまでメソッドはブロックします。False を設定した場合、バックグラウンドで実行されます。loop 引数に True を設定した場合、表示を繰り返します。表示するイメージの色は、color 引数で設定します。color 引数は、タプルまたはリストまたは整数で設定できます。

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

scroll メソッドを実行すると下図のように、LED ディスプレイに赤色で文字'Hello'がずっとスクロール表示されます。

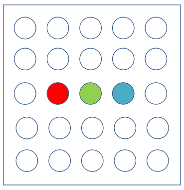


4.1.5. ディスプレイの電源

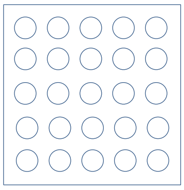
LED ディスプレイは LED の明るさをすべて 0 にした状態でも電気を消費します。off()メソッドで、LED ディスプレイの電源を OFF にできます。また on()メソッドで、LED ディスプレイの電源を ON にできます。LED ディスプレイの電源状態は、is_on()メソッドを使用して確認できます。

```
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()
```

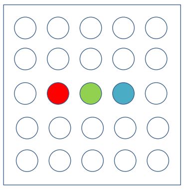
LED ディスプレイには下図のように(1, 2)の位置に赤、(2, 2)の位置に緑、(3, 2)の位置に青が表示され、off メソッド実行に LED が消灯し、on メソッド実行時に LED が点灯します。



off メソッド実行前



off メソッド実行時



on メソッド実行時

変数 ds1 には False が、変数 d2 には True が代入されます。

4.2. StuduinoBitImage クラス

StuduinoBitImage クラスを使用する事で、LED ディスプレイに表示するイメージを簡単に作成することができます。イメージは、LED ディスプレイに表示するパターンと色情報からなります。パターンは、数字の 0(OFF)と 1(ON)を使って表します。下記にイメージと LED ディスプレイでの表示の例を示します。

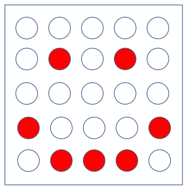
イメージ

■パターン：

```
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
```

■基本の色：赤

LED ディスプレイ



4.2.1. コンストラクタ

４つの方法でイメージを生成できます。

StuduinoBitImage(*,color)	5×5 の空（すべて 0）のイメージを作成します
StuduinoBitImage(string,*,color)	string 引数に 0 と 1 の文字列でパターンを作成します。
StuduinoBitImage(width, height,*, color)	指定された幅(width 引数)と高さ(height 引数)で空のイメージを作成します。
StuduinoBitImage(width, height, buffer,*,color)	指定された幅(width 引数)と高さ(height 引数)で指定されたパターン（buffer 引数）を作成します。

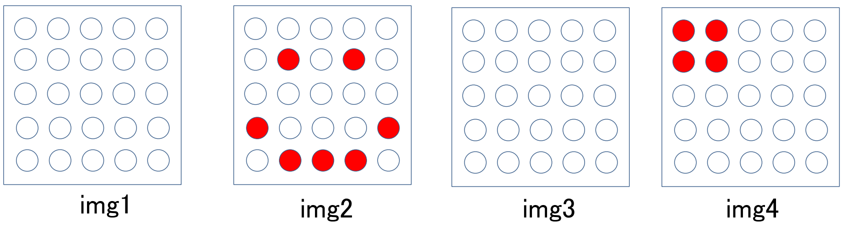
全コンストラクタで、color 引数で基本の色を設定します。color 引数はタプルまたはリストまたは整数で指定します。color 引数が指定されない場合は、赤（(31,0,0)）が基本の色に設定されます。

```
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]))
```

作成したイメージは、下記のようにディスプレイオブジェクトを使用する事で LED ディスプレイに表示できます。

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

変数 img1, img2, img3, img4 を LED ディスプレイに表示すると下図のようになります。



string 引数で設定するコンストラクタは、下記のように 1 行で記述することもできます。

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

4.2.2. イメージのサイズの取得

イメージのサイズの取得には、width()メソッドと heigth()メソッドを使用します。

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

変数 w には 2 が h には 3 が代入されます。

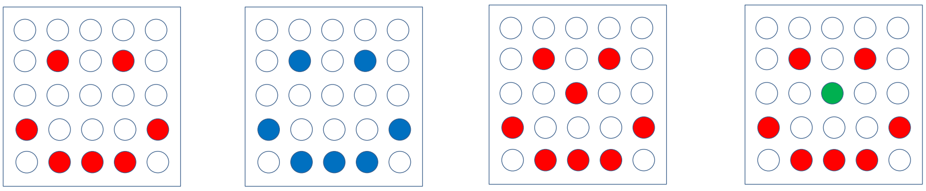
4.2.3. イメージの設定

set_pixel(x, y, value)メソッドで、イメージのパターンを編集できます。設定するピクセルの位置指定は、LED ディスプレイと同様の x-y 座標系（x 引数と y 引数）で指定し、value 引数に 0 または 1 を設定します。また、set_pixel_color(x, y, color)メソッドで、イメージの 1 ピクセルの色を設定できます。設定するピクセルの位置(x 引数と y 引数)と color 引数で色を設定します。

set_base_color(color)メソッドで、基本の色を設定します。

```
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))
```

変数 original, img1, img2, img3 をそれぞれ LED ディスプレイに表示すると、下図のようになります。img1 では set_base_color で基本の色を青色(0, 0, 10)に変更したことが確認できます。img2 では set_pixel で(2, 2)の位置のピクセルを 1 にしたことが確認できます（LED ディスプレイには基本の色で表示されます）。img3 では set_pixel で(2, 2)の位置のピクセルの色を緑色(0, 10, 0)にしたことが確認できます。



original　img1　img2　img3

4.2.4. イメージ情報の取得

get_pixel(x, y)メソッドで、イメージのパターン情報を取得できます。指定したピクセルの位置(x 引数と y 引数)のパターン情報（0 または 1）を返します。

get_pixel_color(x, y)メソッドで、イメージの 1 ピクセルの情報を取得できます。指定したピクセルの位置(x 引数と y 引数)の色情報をタプルで返します。

```
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)
```

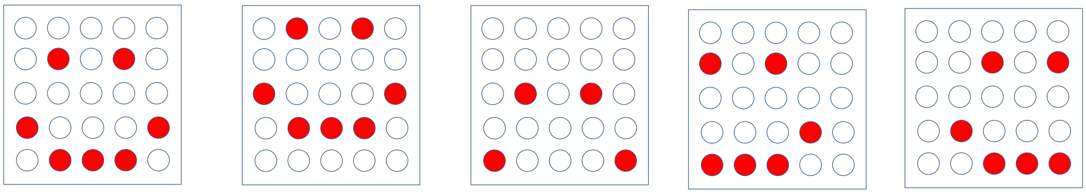
変数 v1 と変数 c1 にはイメージの位置(0, 0)のパターン情報 0 と色情報(0, 0, 0)が、変数 v2 と変数 c2 には位置(1, 1)のパターン情報 1 と色情報 (31, 0, 0)がそれぞれ代入されます。

4.2.5. イメージの操作

shift_up(n)メソッド、shift_down(n)メソッド、shift_left(n)メソッド、shift_right(n)メソッドを使用して、n 引数分の上下左右に移動したイメージを作成できます。

```
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)
```

変数 original, up, down, left, right をそれぞれ LED ディスプレイに表示すると、下図のようになります。original に対して 1 つずつ上下左右に移動していることが確認できます。

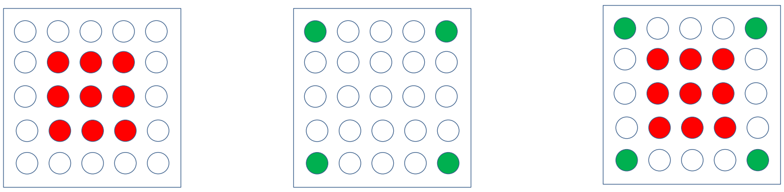


original　up　down　left　right

+演算子を使用する事で、複数のイメージを合わせたイメージを作成できます。

```
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
```

変数 img1, img2, img3 をそれぞれ LED ディスプレイに表示すると、下図のようになります。img1 と img2 が img3 に統合されていることが確認できます。



img1　img2　img3

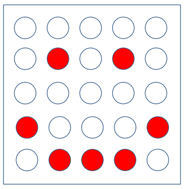
copy()メソッドを使用して、イメージを複製できます。

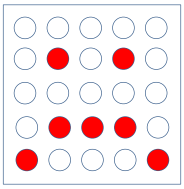
4.2.6. 組み込みイメージ

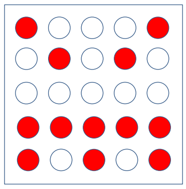
StuduinoBitImage であらかじめ用意されたパターンを使用することができます。

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

変数 img1, img2, img3 をそれぞれ LED ディスプレイに表示すると、下図のようになります。


img1


img2


img3

StuduinoBitImage で提供する全組み込みイメージは、5.3 を参照してください。

4.3. StuduinoBitBuzzer クラス

StuduinoBitBuzzer クラスは、ブザーを制御するクラスです。

4.3.1. コンストラクタ

ブザーを制御するオブジェクトを作成します。

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

4.3.2. 音の制御

on(sound, *, duration=None)メソッドで、ブザーから音を出します。sound 引数に、MIDI ノートナンバーまたは音名または周波数を設定します。指定可能なノートナンバーと音名は、5.2 を参照してください。duration 引数に出力する音の長さをミリ秒単位で設定します。duration 引数を設定しない場合、off()メソッドで音を止めます。

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

ブザーから'C4'の音が 1 秒間出ます。

4.3.3. PWM の解放

音の出力には PWM を使用しています。Studuino:bit では、同時に使用できる PWM は 4 つまでです。PWM が足りず、作成したブザーオブジェクトを使用しない場合、release()メソッドをコールしてブザーオブジェクトに割り当てられた PWM を解放してください。

4.4. StuduinoBitButton クラス

StuduinoBitButton クラスは、基板の前面にある 2 つのボタンを制御するクラスです。

4.4.1. コンストラクタ

ボタンを制御するオブジェクトを作成します。引数に'A'または'B'を設定してください。

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

4.4.2. ボタンの状態取得

get_value()メソッドで、ボタンの値を 0 / 1 の整数で取得します。is_pressed()メソッドで、ボタンが押されている状態を True / False で取得します。was_pressed()で、ボタンが過去に押されたかどうかをを True / False で取得します。get_presses()メソッドで、過去にボタンが何回押されたかを取得します。

```
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')
```

A ボタンが押されたタイミングで"A button is pressed"が標準出力に表示されます。

4.5. StuduinoBitLightSensor クラス

StuduinoBitLightSensor クラスは、基板の光センサーを制御するクラスです。

4.5.1. コンストラクタ

光センサーを制御するオブジェクトを生成します。

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

4.5.2. 明るさの取得

get_value()メソッドで、光センサーのアナログ値を取得します。センサーの値は 0 から 4095 の整数です。

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

500 ミリ秒ごとに光センサーの値が標準出力に表示されます。

4.6. StuduinoBitTemperature クラス

StuduinoBitTemperature クラスは、基板の温度センサーを制御するクラスです。

4.6.1. コンストラクタ

温度センサーを制御するオブジェクトを生成します。

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

4.6.2. 温度の取得

get_value()メソッドで、温度センサーのアナログ値を取得します。センサーの値は 0 から 4095 の整数です。get_celsius(ndigits=2)メソッドで温度センサーから温度の値をセ氏（℃）で取得します。ndigits 引数で小数点以下の桁数を設定します。

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

500 ミリ秒ごとに温度センサーのアナログ値と温度の値が標準出力に表示されます。

4.7. StuduinoBitAccelerometer クラス

StuduinoBitAccelerometer クラスは、基板の加速度センサーを制御するクラスです。

4.7.1. コンストラクタ

加速度センサーを制御するオブジェクトを作成します。

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

StuduinoBitAccelerometer のコンストラクタは、fs 引数にフルスケール（最大測定限界）と sf 引数に単位を設定することができます。fs 引数には、'2g'、'4g'、'8g'、'16g'のいずれかを設定でき、それぞれ重力加速度 2G、4G、8G、16G まで測定できることを表します。デフォルトは、2G になります。sf 引数には、'ms2'、'mg'のどちらかを設定でき、それぞれ m/秒^2、ミリ G を表します。デフォルトは、m/秒^2 です。

4.7.2. 加速度センサーの設定

set_fs(value)メソッドで、フルスケール（最大測定限界）を設定することができます。value 引数には、'2g'、'4g'、'8g'、'16g'のいずれかを設定できます。set_sf(value)メソッドで、単位を設定することができます。'ms2'、'mg'のどちらかを設定できます。

4.7.3. 加速度の取得

get_values(ndigits=2)メソッドで、加速度センサーの値を取得します。センサーの値は、タプル(x, y, z)で取得します。get_x(ndigits=2)メソッド、get_y(ndigits=2)メソッド、get_z(ndigits=2)メソッドで x 軸方向、y 軸方向、z 軸方向の加速度を別々に取得することができます。いずれのメソッドも ndigits 引数で必要な小数点以下の桁数を設定します。

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

500 ミリ秒ごとに加速度センサーの値が標準出力に表示されます。

4.8. StuduinoBitGyro クラス

StuduinoBitGyro クラスは、基板のジャイロセンサーを制御するクラスです。

4.8.1. コンストラクタ

ジャイロセンサーを制御するオブジェクトを作成します。

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

StuduinoBitGyro のコンストラクタは、fs 引数にフルスケール（最大測定限界）と sf 引数に単位を設定することができます。fs 引数には、'250dps'、' 500dps'、' 1000dps'、' 2000dps' のいずれかを設定でき、それぞれ角速度 250 度/秒、500 度/秒、1000 度/秒、2000 度/秒まで測定できることを表します。デフォルトは、250 度/秒になります。sf 引数には、'dps'、'rps' のどちらかを設定でき、それぞれ度/秒、ラジアン/秒を表します。デフォルトは、度/秒です。

4.8.2. ジャイロセンサーの設定

set_fs(value)メソッドで、フルスケール（最大測定限界）を設定することができます。value 引数には、'250dps'、'500dps'、'1000dps'、'2000dps'のいずれかを設定できます。set_sf(value) メソッドで、単位を設定することができます。'dps'、'rps'のどちらかを設定できます。

4.8.3. 角速度の取得

get_values(ndigits=2)メソッドで、ジャイロセンサーの値を取得します。センサーの値は、コンストラクタや set_sf メソッドで設定した単位で、タプル(x, y, z)で取得します。get_x(ndigits=2)メソッド、get_y(ndigits=2)メソッド、get_z(ndigits=2)メソッドで x 軸方向、

y 軸方向、z 軸方向の角速度を別々に取得することができます。いずれのメソッドも ndigits 引数で必要な小数点以下の桁数を設定します。

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

500 ミリ秒ごとにジャイロセンサーの値が標準出力に表示されます。

4.9. StuduinoBitCompass クラス

StuduinoBitCompass クラスは、基板の地磁気センサーを制御するクラスです。

4.9.1. コンストラクタ

地磁気センサーを制御するオブジェクトを作成します。

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

4.9.2. キャリブレーション

calibrate()メソッドで、地磁気センサーのキャリブレーションができます。キャリブレーションには LED ディスプレイを使用します。

is_calibrated() メソッドで、キャリブレーション済みかどうかを確認できます。

clear_calibration()メソッドで、キャリブレーション情報をクリアできます。

4.9.3. 方角の取得

heading()メソッドで、方角情報（0～360 度の角度）を得ることができます。地磁気センサーのキャリブレーションがすんでいない場合、heading()メソッド実行時に自動的にキャリブレーションが開始されます。Studuino:bit の LED ディスプレイを上に向けた状態で、USB コネクタ側が真南を向いているときが 0 度で、右回りに角度は増えていきます。

4.9.4. 地磁気センサーの値の取得

get_values()メソッドで、地磁気センサーの値を取得します。センサーの値は、μT(マイクロ・テスラ)で、タプル(x, y, z)で取得します。

get_x()メソッド、get_y()メソッド、get_z()メソッドで x 軸方向、y 軸方向、z 軸方向のセンサーの値を別々に取得することができます。

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

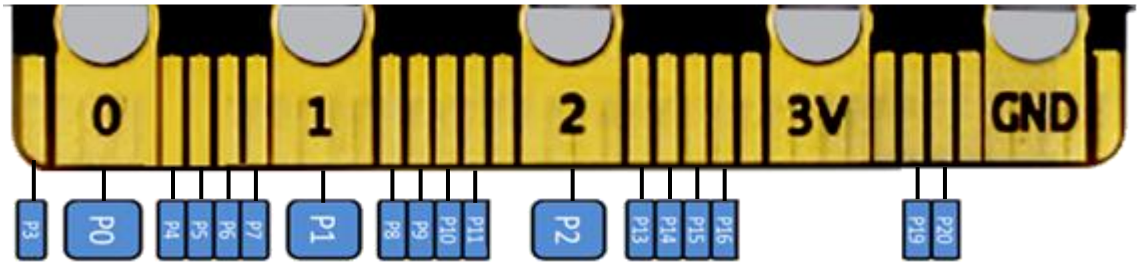
500 ミリ秒ごとに地磁気センサーの値が標準出力に表示されます。

4.10. StuduinoBitTerminal クラス

StuduinoBitTerminal クラスは、基板の端子を制御するクラスです。

4.10.1. コンストラクタ

StuduinoBitTerminal のコンストラクタで、P17 と P18 を除く P0 から P20 の端子を制御するオブジェクトを作成します。StuduinoBitTerminal クラスで制御できる Studuino:bit の端子は、下図の P0～P20 です。



P0～P20 を指定して、端子を制御するオブジェクトを作成します。

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

P0～P20 端子と入出力の種類を下記に記します。

端子	入出力の種類	端子	入出力の種類
P0	デジタル入出力・アナログ入力	P10	デジタル入出力
P1	デジタル入出力・アナログ入力	P11	デジタル入出力
P2	デジタル入力・アナログ入力	P12	デジタル入出力
P3	デジタル入力・アナログ入力	P13	デジタル入出力
P4	デジタル入出力	P14	デジタル入出力
P5	デジタル入出力	P15	デジタル入出力
P6	デジタル入出力	P16	デジタル入出力
P7	デジタル入出力	P19	デジタル入出力

P8	デジタル入出力	P20	デジタル入出力
P9	デジタル入出力		

4.10.2. デジタル入出力

作成する端子オブジェクトの入出力の種類がデジタル入出力の場合、write_digital(value)メソッドで、デジタル信号を出力します。value 引数には、0 または 1 を設定します。また、read_digial()メソッドで、デジタル信号を読み取ります。PWM 出力に使用する周波数をset_analog_hz()メソッドで設定し、write_analog(value)メソッドで PWM 信号を出力します。

4.10.3. デジタル入出力・アナログ入力

作成する端子オブジェクトの入出力の種類がデジタル・アナログ入出力の場合、デジタル入出力のメソッドに加えて、read_analog()メソッドで、アナログ信号を読み取ることができます。

4.10.4. デジタル入力・アナログ入力

作成する端子オブジェクトの入出力の種類がデジタル入力・アナログ入力の場合、read_digital()メソッドと read_analog()メソッドのみ使用できます。

4.11. StuduinoBitRadio クラス

StuduinoBitRadio クラスは、基板の Wi-Fi を使用して基板間通信を制御するクラスです。

4.11.1. コンストラクタ

基板間通信を制御するオブジェクトを作成します。

```
from pystubit.nw import StuduinoBitRadio
radio = StuduinoBitRadio()
```

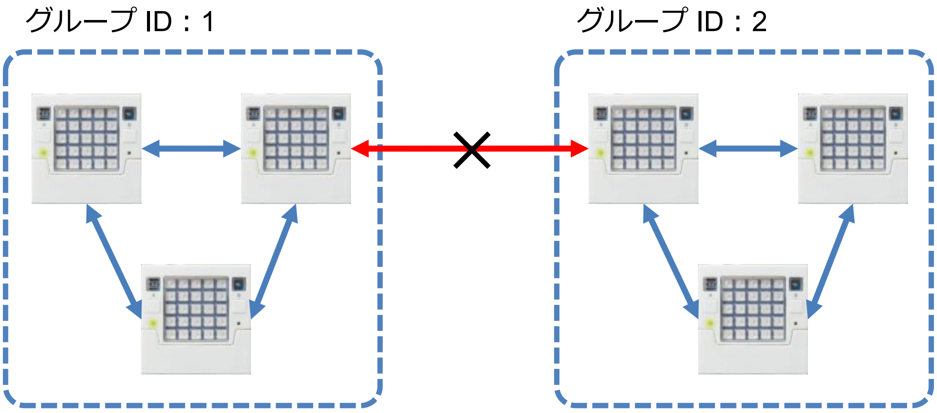
4.11.2. 基板間通信の ON/OFF

on()メソッドで、基板間通信機能を ON にします。off()メソッドで、基板間通信機能を OFF にします。基板間通信を行う前には、必ず on()メソッドで、基板間通信機能を ON にしてください。

4.11.3. 基板間通信の開始とグループ

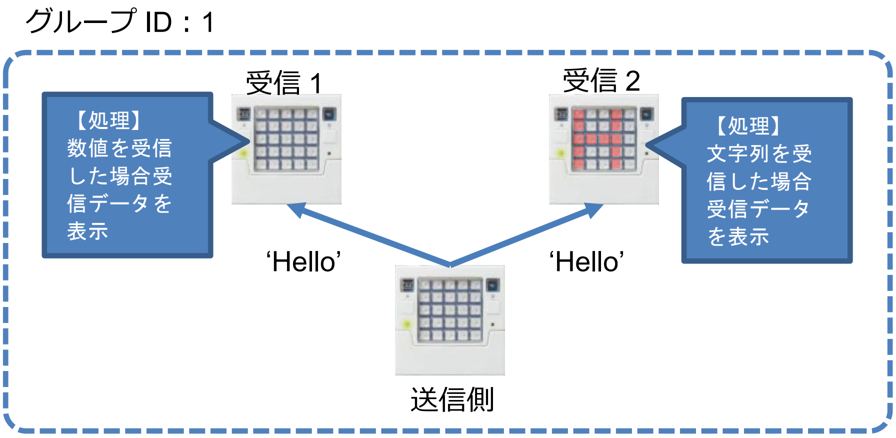
start(group)メソッドで、基板間通信を開始します。group 引数には基板が所属する通信グループ ID（0～255）を設定します。下図のように group 引数で指定した ID が同じ基板間では双方向の通信が行えますが、ID が異なる基板間では通信ができません。

基板間通信開始後にグループ ID を変更する場合は、group(group)メソッドを使用します。



4.11.4. 送信・受信

基板間通信では、送信側は同グループに所属している基板にブロードキャストで送信します。受信側では必要なデータを受信して処理を行います。下の図のように、送信側はデータ（'Hello'という文字列）をグループに属するすべての基板にブロードキャスト送信し、受信側では受信したデータに対する受信処理を行います。



送信メソッドは、send_number(n)、send_value(s, n)、send_string(s)、send_buffer(buf)です。send_number メソッドは、n 引数で設定された数値を送信します。send_value メソッドは、s 引数で設定された文字列と n 引数で設定された数値のペアを送信します。send_string メソッドは、s 引数で設定された文字列を送信します。send_buffer メソッドは、buf 引数で設定されたバイト列を送信します。

受信メソッドは、recv()です。recv()メソッドは、send_*メソッドで送信されたデータを受信し、受信したデータを戻り値で返します。recv()メソッドはノンブロッキングで、受信するデータがない場合は、None を返します。データを受信した場合、戻り値には受信したデータのタイプを表す ID とデータのタプル(id, data)が返ります。ID は、0 が数値、1 が変数、

2 が文字列、3 がバイト列になります。

上の図で示した基板 3 台を使用した基板間通信のサンプルプログラムを以下に示します。

【送信側】

```
from pystubit.nw import StuduinoBitRadio
import time

send = StuduinoBitRadio()
send.on()
send.start(0)

while True:
    send.send_string( ‘Hello’ )
    time.sleep(1)
```

送信側のプログラムでは、グループ 0 に文字列’Hello’を 1 秒間隔で送信しています。

【受信 1】

```
from pystubit.nw import StuduinoBitRadio
from pystubit.board import display

recv1 = StuduinoBitRadio()
recv1.on()
recv1.start(0)

while True:
    data = recv1.recv()
    if not data is None:
        if (data[0] == 0):
            display.show(str(data[1]))
```

受信 1 のプログラムでは、数値を受信した時のみ LED ディスプレイに受信した数字を表示します。

【受信 2】

```
from pystubit.nw import StuduinoBitRadio
from pystubit.board import display

recv2 = StuduinoBitRadio()
recv2.on()
recv2.start(0)

while True:
	data = recv2.recv()
	if not data is None:
		if (data[0] == 2):
			display.show(data[1])
```

受信の 2 プログラムでは、文字列を受信した時のみ LED ディスプレイに受信した文字列を表示します。

5. 付録

5.1. board モジュール

board モジュールには、Studuino:bit のハードウェアを制御するオブジェクトが定義されています。下記のように宣言することで、オブジェクトを使用できます。

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

board モジュールに定義されているオブジェクトを下記に記します。

オブジェクト	ハードウェア
display	LED ディスプレイ
Image（クラス）	LED ディスプレイ（イメージ）
buzzer	電子ブザー
button_a, button_b	ボタン A，ボタン B
lightsensor	光センサー
temperature	温度センサー
accelerometer	加速度センサー
gyro	ジャイロセンサー
compass	地磁気センサー
p0～p16, p19, p20	エッジコネクター

pin0～pin3	micro:bit 互換エッジコネクター（※）

※：pin0 と p0、pin1 と p1、pin2 と p2、pin3 と p3 の併用はできません。

※：pin0～pin3 オブジェクトは、p0～p3 オブジェクトと同じメソッドが使用できますが、read_analog()メソッドは、0～1023 の値を返します。

5.2. 音について

StuduinoBitBuzzer クラスで使用できる MIDI ノート番号と音名です。

MIDI	音名	MIDI	音名	MIDI	音名	MIDI	音名	MIDI	音名	MIDI	音名	MIDI	音名
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. 組み込みイメージ

組み込みイメージの一覧を下記に記します。

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. クラス一覧

以下に、各クラスのメソッド一覧を記し、board モジュールを使用した場合のオブジェクトとの対応を記します。

機能	クラス	board モジュール	メソッド	説明
ボタン	StuduinoBitButton	button_a button_b	_init_(ab)	ボタン A/B のオブジェクトを作成します。引数 ab には'A'または'B'を指定します。
			get_value()	ボタンが押されている場合 0 を、押されていない場合 1 を返します。
			is_pressed()	ボタンが押されている場合 True を返します。
			was_pressed()	ボタンオブジェクトは、ボタンが押された情報を内部にもっています。本メソッドは、ボタンが過去に押されていたら True を返します。コール時に情報をリセットします。
			get_presses()	ボタンオブジェクトは、ボタンが押された情報を内部にもっています。本メソッドは、ボタンが押された回数を返します。コール時に回数をリセットします。
LED ディスプレイ (フルカラー)	StuduinoBitDispaly	display	_init_()	ディスプレイオブジェクトを生成します。
			get_pixel(x, y)	x 列 y 行の LED の色を返します。返り値は(R,G,B)です。
			set_pixel(x, y, color)	x 列 y 行の LED の色を color で設定します。color は、(R,G,B), [R,G,B], #RGB で指定します。
			clear()	すべての LED の明るさを 0 (オフ)に設定します。
			show(iterable, delay=400, *, wait=True, loop=False, clear=False, color=None)	iterable(イメージ/文字列/数字)を順番に表示します。
			scroll(string, delay=150, *, wait=True, loop=False, color=None)	value(文字/数字) を水平方向にスクロールさせます。
			on()	ディスプレイの電源を ON にします。
			off()	ディスプレイの電源を OFF にします。(ディスプレイに関連づけられた GPIO 端子を他の目的に再利用できるようにします)。

			is_on()	ディスプレイの電源が ON であれば True を、OFF であれば False を返します。
イメージ	StuduinoBitImage	Image	__init__(string, color=None) __init__(width=None, height=None, buffer=None, , color=None)	string 引数に 0:OFF, 1:ON として LED ディスプレイに表示するパターンを文字列で設定しイメージオブジェクトを作成します。 StuduinoBitImage('01100:10010:11110:10010:10010:', color=(0,0,10)) または、width 引数と height 引数で幅と高さを指定しイメージオブジェクトを作成します。 StuduinoBitImage(2, 2,bytearray([0,1,0,1]) StuduinoBitImage(3, 3) micro:bit と同様、0-9 の値を指定することもできますが、0 が OFF, 1-9 が ON として使用されます。color 引数を指定しない場合は(RGB)=(31,0,0)が設定されます。
			width()	イメージの列数を返します。
			height()	イメージの行数を返します。
			set_pixel(x, y, value)	イメージの位置(x, y)のピクセルの値を value 引数で設定します。value 引数には、0(OFF)または 1(ON)を設定します。
			set_pixel_color(x, y, color)	イメージの位置(x, y)のピクセルの色を color 引数に設定します。color 引数には(R, G, B)または、[R, G, B]または、#RGB を設定します。
			get_pixel(x, y)	イメージの(x, y)の位置のピクセルの値(1 または 0)を返します。
			get_pixel_color(x, y, hex=False)	イメージの(x, y)の位置のピクセルの色を返します。 hex 引数に False を設定した場合、(R,G,B)を返します。True を設定した場合、#RGB を返します。
			set_base_color(self, color)	全ピクセルの色を color 引数で指定した色に設定します。color 引数には(R, G, B)または、[R, G, B]または、#RGB を設定します。
			shift_left(n)	n 引数に指定した分だけ左にシフトした新しいイメージを返します。
			shift_right(n)	shift_left(-n) と同じです。
			shift_up(n)	n 引数に指定した分だけ上にシフトした新しいイメージを返します。
			shift_down(n)	shift_up(-n) と同じです。
			copy()	イメージ全体のコピーを返します。
			repr(image)	イメージのコンパクトな文字列表現を得ます。
			str(image)	イメージの可読可能な文字列表現を得ます。
			+	2つのイメージの各ピクセルを足した新しいイメージを作成します。

				組み込みのイメージ（5.3 を参照）
端子	StuduinoBitTreminal	端子モジュールは下記になります	_init_(pin)	pin 引数に StuduinoBit の端子を表す文字列'P0'～'P16', 'P19', 'P20'のいずれかを指定して端子オブジェクトを作成します。
デジタル入出力 アナログ入力（※）	StuduinoBitAnalogDigitalPin	p0,p1,p2,p3 【注】 p2,p3 はデジタル入出力ができません。write_digital メソッドを実行するとエラーになります。また、read_digital メソッドは、アナログ入力で実現しているため使用できます。	write_digital(value)	value 引数が 1 の場合は High に、0 の場合は Low にデジタル信号を設定します。
			read_digital()	デジタル信号を取得します。High の場合は 1 を、Low の場合は 0 を返します。
			write_analog(value)	PWM 信号を端子に出力します。value 引数は、0(0%)～1023(100%)です。
			set_analog_period(period, timer=-1)	PWM 信号の周期を period にミリ秒単位で設定します。
			set_analog_period_microseconds(period, timer=-1)	PWM 信号の周期を period にマイクロ秒単位で設定します。
			set_analog_hz(hz, timer=-1)	PWM 信号の周期を周波数で設定します。
			status()	PWM の使用状況を表示します
			read_analog(mv=False)	端子の電圧を読み取り、mv 引数が False の場合、0（0V）から 4096（3.3V）までの間の整数値を返します。mv=True の場合、mV で返します。
デジタル入出力 （※）	StuduinoBitDigitalPin	p4～P16,p19,p20	write_digital(value)	value 引数が 1 の場合は High に、0 の場合は Low にデジタル信号を設定します。
			read_digital()	デジタル信号を取得します。High の場合は 1 を、Low の場合は 0 を返します。
			write_analog(value)	PWM 信号を端子に出力します。value 引数は、0(0%)～1023(100%)です。
			set_analog_period(period, timer=-1)	PWM 信号の周期を period にミリ秒単位で設定します。
			set_analog_period_microseconds(period, timer=-1)	PWM 信号の周期を period にマイクロ秒単位で設定します。
			set_analog_hz(hz, timer=-1)	PWM 信号の周期を周波数で設定します。
			status()	PWM の使用状況を表示します
ブザー	StuduinoBitBuzzer	buzzer	_init_()	ブザーオブジェクトを作成します。

			on(sound, *, duration=-1)	sound 引数に音名(‘C3’ ～’ G9’ の文字列)、または MIDI ノートナンバー(‘48’ ～’ 127’ の文字列)、または周波数(整数)を、duration 引数に音の出力時間(ms)を指定して、ブザーを制御します。duration 引数を指定しない場合は、off メソッドがコールされるまで音を出力します。 bzr.on('50', duration=1000)　　# MIDI ノートナンバーの 50 番の高さの音が 1 秒間出力されます。 bzr.on('C4', duration=1000)　　#　音名の C4 の高さの音が 1 秒間出力されます。 bzr.on(440)
			off()	ブザーから出力している音を停止します。
温度センサー	StuduinoBitTemperature	temperature	_init_()	温度センサーオブジェクトを作成します。
			get_value()	基板上の温度センサーの値(0-4095)を返します。
			get_celsius()	基板上の温度センサーの値を温度(セ氏)を返します
光センサー	StuduinoBitLightSensor	lightsensor	_init_()	光センサーオブジェクトを作成します。
			get_value()	基板上の光センサーの値(0-4095)を返します。
加速度センサー	StuduinoBitAccelerometer	accelerometer	_init_(fs=’ 2G’ , sf=’ mg2’)	fs 引数に最大測定限界を、'2g'/'4g'/'8g'/'16g'で、sf 引数に単位を 'mg'/'ms2'で指定して加速度センサーオブジェクトを作成します。 デフォルトは fs=’ 2G’ 、sf=’ ms2’ になります。
			get_x	x 軸の加速度を計測します。
			get_y	y 軸の加速度を計測します。
			get_z	z 軸の加速度を計測します。
			get_values()	=(get_x(), get_y(), get_z())
			set_axis(mode)	mode には、次の文字列'sbmp'(Studuino:bit MicroPython)、'sbs'(Studino:bit Software)、'mb'(micro:bit)のいずれかを設定します。デフォルトは'sbmp'です。
			set_fs(value)	センサーの最大測定限界を’ 2g'/'4g'/'8g'/'16g'で設定します。
			set_sf(value)	センサーの単位を'mg'/'ms2'で設定します。
ジャイロセンサー	StuduinoBitGyro	gyro	_init_(fs=’ 250dps’ , sf=’ dps’)	fs 引数に最大測定限界を、'250dps'/'500dps'/'1000dps'/'2000dps'で、sf 引数に単位を'dps'/'rps'で指定してジャイロセンサーオブジェクトを作成します。 デフォルトは fs=’ 250dps’ 、sf=’ dps’ になります
			get_x	x 軸の角速度を計測します。
			get_y	y 軸の角速度を計測します。

			get_z	z 軸の角速度を計測します。
			get_values()	=(get_x(), get_y(), get_z())
			set_axis(mode)	mode には、次の文字列'sbmp'(Studuino:bit MicroPython)、'sbs'(Studino:bit Software)、'mb'(micro:bit)のいずれかを設定します。デフォルトは'sbmp'です。
			set_fs(value)	センサーの最大測定限界を'250dps'/'500dps'/'1000dps'/'2000dps'で設定します。
			set_sf(value)	センサーの単位を'dps'/'rps'で設定します。
コンパス	StuduinoBitCompass	compass	_init_()	コンパスのオブジェクトを作成します。最大測定限界は、±4500 μT で単位は μT(マイクロ・テスラ)です。
			get_x	x 軸の磁場の強さを返します。
			get_y	y 軸の磁場の強さを返します。
			get_z	z 軸の磁場の強さを返します。
			get_values()	=(get_x(), get_y(), get_z())
			set_axis(mode)	mode には、次の文字列'sbmp'(Studuino:bit MicroPython)、'sbs'(Studino:bit Software)、'mb'(micro:bit)のいずれかを設定します。デフォルトは'sbmp'です。
			calibrate()	磁気センサーのキャリブレーションを開始します。LED ディスプレイ上に円を描くようデバイスを回転させます。
			is_calibrated()	キャリブレーション済の場合は True、そうでない場合は False を返します。
			clear_calibration()	キャリブレーションを情報を削除します。
			heading()	磁針の向きを返します。0～360° の整数で、時計回り、北が 180°
BLE	StuduinoBitBLE			T.B.D
無線通信	StuduinoBitRadio	オブジェクトはありません。	_init_()	無線通信オブジェクトを作成します。
			on()	無線通信をオンにします。無線通信は電力を消費し、他で必要となるかもしれないメモリも占有するため、明示的に呼び出す必要があります。
			off()	無線通信をオフにして、電力とメモリを節約します。
			無線通信がオンになるまで、以下の送受信のメソッドは機能しません。	
			start(group)	group 引数に 0～255 の値でグループを設定し、通信開始状態にします。
			send_number(n)	値を送信します。値は、32bit 整数です。
			send_value(s, n)	値と変数名を送信します。値は、32bit 整数で変数名は 13 文字までです。
			send_string(s)	文字列を送信します。文字列は、19 文字までです。

			send_buffer(buf)	バイト列を送信します。19 バイトまでです。
			recv()	データを受信します。受信するデータがない場合は、None を返します。
			group(group=-1)	group 引数に 0～255 の値でグループを設定します。
WiFi	StuduinoBitWiFi			T.B.D

（※）デジタル入出力、アナログ入力、アナログデジタル入出力のクラスは、単体でインスタンス化されることは想定しておりません。端子クラスを通してインスタンス化されることを想定しています。

6. お問い合わせ先

株式会社 アーテック お客様相談窓口

お電話によるお問い合わせ **072-990-5656**

お問い合わせフォーム **https://www.artec-kk.co.jp/contact/**