

Qiita Advent Calendar 2021 を見てみよう！新着カレンダーはこちら

詳しくはこちら

⚠ この記事は最終更新日から1年以上が経過しています。

@c60evaporator

投稿日 2020年05月08日 更新日 2020年07月21日

Omron環境センサの値をRaspberryPiで定期ロギングする

Python, GoogleAppsScript, RaspberryPi, bluetooth

環境センサとは？

Omronの環境センサは、温度、湿度、照度、気圧、騒音など、複数のセンサーをひとまとめにしたデバイスで、センサメーカーの大御所らしい充実した機能が魅力です！



(左：BAG型の2JCIE-BL01、 右：USB型の2JCIE-BU01)

本記事では、RaspberryPiから環境センサの値を取得し、Googleスプレッドシートにアップロードしてみました。

実際に試してみて、センシングからロギングまで、
安価な汎用機器だけでここまで機能が実現できるのか！
と驚いています。
(…その代わり、心が折れそうになるハマりポイントも多いですが笑)

必要なもの

- **RaspberryPi**（今回はPi3Model Bを使用）
- **Python実行環境**（今回はpyenvでPython3.7.6使用）
- **Googleアカウント**（スプレッドシートを使うのに必要）
- **スマホ**（BLE Scanner設定用）
- **Omron 環境センサ（BAG型）** ※USB型は別途記事作成します

手順

- ① RaspberryPiとセンサのBluetooth接続確認
- ② 環境センサのモードをBroadcasterに変更
- ③ 環境センサの測定値をPythonで取得
- ④ PythonからGASのAPIを叩いてスプレッドシートにデータ書き込み
- ⑤ スクリプトの定期実行

基本的にはRaspberryPiが処理の起点（セントラル）となります。

※参考にさせて頂いた記事

- ・センサからのデータ取得

<https://ambidata.io/samples/temphumid/ble-gw-omron/>

<https://homemadegarbage.com/gobernables>

- ・Googleスプレッドシートへのアップロード

<https://akerun.hateblo.jp/entry/2018/12/07/094714>

方法

① RaspberryPiとセンサのBluetooth接続確認

環境センサの認識確認

- ・環境センサの動作確認

電池を入れ、ランプが一瞬点くことを確認してください

- ・Bluetooth機器のスキャン

RaspberryPiで下記コマンドを実行

```
sudo hcitool lescan
```

```
LE Scan ...  
DD:CC:BB:AA:77:66 Env
```

というように、「Env」という名前がでてきたら、これが環境センサのMACアドレスです。

出てこなければ電池の接触やRaspberryPiのBluetooth有効を確認してください。

bluepyでの認識確認

bluepyは、PythonでBluetooth Low Energy(BLE)にアクセスするためのライブラリです（[クラス定義](#)）

ここでは、bluepyで環境センサを認識できるかを確認します。

- ・必要なパッケージのインストール

下記をインストールします

```
sudo apt install libglib2.0-dev
```

・bluepyのインストール

下記コマンドでpipでインストールします

```
pip install bluepy
```

・bluepyに権限を付与

スキャンにはbluepyにSudo権限を与える必要があります。

bluepyのインストールされているフォルダに移動し、

```
cd ~/.pyenv/versions/3.7.6/lib/python3.7/site-packages/bluepy
```

※上記は、pyenvでPython3.7.6をインストールした場合。

環境により場所は異なるので注意

デフォルトのPython環境の場合pip3 show bluepy"でインストール場所を調べられる

下記コマンドでbluepy-helperにSudo権限を付与する

```
sudo setcap 'cap_net_raw,cap_net_admin+eip' bluepy-helper
```

・スキャン用コードの作成

下記コードを作成します

```
ble_scan.py

from bluepy import btle

class ScanDelegate(btle.DefaultDelegate):
    def __init__(self): # コンストラクタ
        btle.DefaultDelegate.__init__(self)

    def handleDiscovery(self, dev, isNewDev, isNewData): # スキャンハンドラー
        if isNewDev: # 新しいデバイスが見つかったら
            print('found dev (%s)' % dev.addr)

scanner = btle.Scanner().withDelegate(ScanDelegate())
while True:
    scanner.scan(10.0)
```

上記コードの動作ですが、

肝となっているのは

Scannerクラス

DefaultDelegateクラス

ScanEntryクラス

です。

上記コードでは

- ・**Scanner.scanメソッド**：引数(上記の場合10秒)をタイムアウト値とし、Scanner.withDelegateメソッドで引数指定したデリゲートを実行する

- ・**DefaultDelegateクラス** : Scanner.withDelegateの引数用デリゲートを作成するための継承用クラス。handleDiscoveryメソッド内に具体的な処理を記述する
- ・**ScanEntryクラス** : Scanner.scanで取得したデバイス情報を保持。handleDiscoveryの引数「dev」が相当を使用し、

「10秒をタイムアウト値とし、新たに見つかったデバイスのMacアドレスを表示し続ける」という処理を実現しています。

・スキャン用コードの実行

```
python ble_scan.py
```

下記のように、LE Scanのときと同じMACアドレスが出てくれば成功です
(MACアドレスが小文字なので注意してください)

```
found dev (dd:cc:bb:aa:77:66)
```

以上で、bluepyでの環境センサ認識を確認できました。

②環境センサのモードをBroadcasterに変更

こちらの記事に記載されているように、
BLEデバイスのデータ送信（アドバタイズ）には
・コネクトモード：双方向通信でデータを取得
・ブロードキャストモード（Broadcaster）：デバイス→セントラルに一方的にデータ送信
の2種類があり、
消費電力面でBroadcasterが有利だそうです。
(逆にコネクトモードはデバイス設定を変える複雑な動作が可能)

電池式の環境センサの稼働時間を上げるために、
Broadcasterに変更して以下の操作を進めます

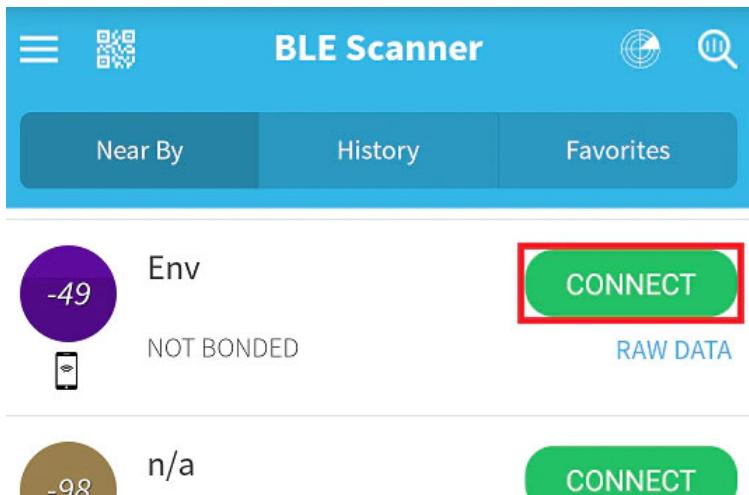
BLE Scannerのインストール

モード変更に使用するBLE Scannearというツールを、
下記からスマートフォンにインストールしてください
Android
iOS

環境センサのアドバタイズ設定変更

アドバタイズ設定を変更し、モードをBroadcasterにします。
アドバタイズ送信間隔もここで変更できます（スピードが必要な用途向け）

- ・BLE Scannerを開き、「Env」と表示されたデバイスにCONNECT



- ・"0C4C3042～"がアドバタイズ設定なので、探す

CUSTOM SERVICE

- 0C4C3010-7700-46F4-AA96-D5E974E32A54
PRIMARY SERVICE

CUSTOM SERVICE

- 0C4C3030-7700-46F4-AA96-D5E974E32A54
PRIMARY SERVICE

CUSTOM SERVICE

- 0C4C3040-7700-46F4-AA96-D5E974E32A54
PRIMARY SERVICE

CUSTOM SERVICE

- 0C4C3040-7700-46F4-AA96-D5E974E32A54
PRIMARY SERVICE

CUSTOM CHARACTERISTIC

R W

UUID: 0C4C3041-7700-46F4-AA96-D5E974E32A54

Properties: READ,WRITE

Value: L0wF

Hex:

0x0C4C3000770046F4AA96D5E974E32A540000000000

Write Type: WRITE REQUEST

③

CUSTOM CHARACTERISTIC

① R W

UUID: 0C4C3042-7700-46F4-AA96-D5E974E32A54

Properties: READ,WRITE

Value:

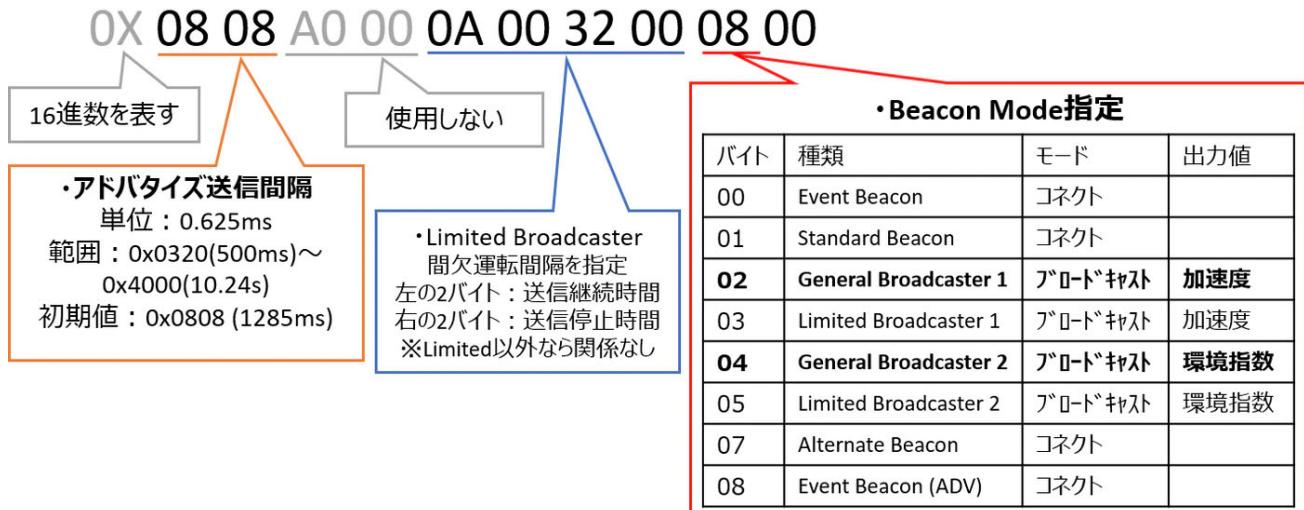
Hex: 0x0808A0000A0032000800

② 内容確認

Write Type: WRITE REQUEST

- ・上図②のアドバタイズ設定の見方

オムロン公式ユーザーズマニュアルを読み解くと、下図のようになっています。



・ブロードキャストモードに変更したい場合 : Beacon Modeを02か04に

・更新間隔を短くしたい場合 : アドバタイズ送信間隔を小さく

という変更が必要となります。

※ 「Limited Broadcaster」とは? と思われるかもしれません、恐らく電池節約のために間欠的にアドバタイズする設定と思われます。認識できなくなるリスクが記載されているので、今回は使用しません。

Beacon Modeは02か04かで迷うところですが、

今回は環境指数を重視して、

Before : 0808A0000A003200 **08** 00

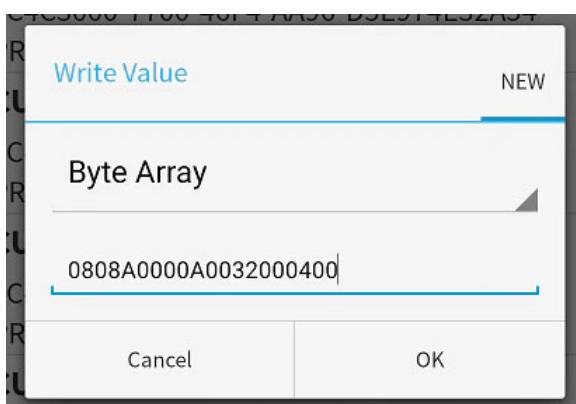
After : 0808A0000A003200 **04** 00

とします。

(加速度センサを使用する場合は02として下さい)

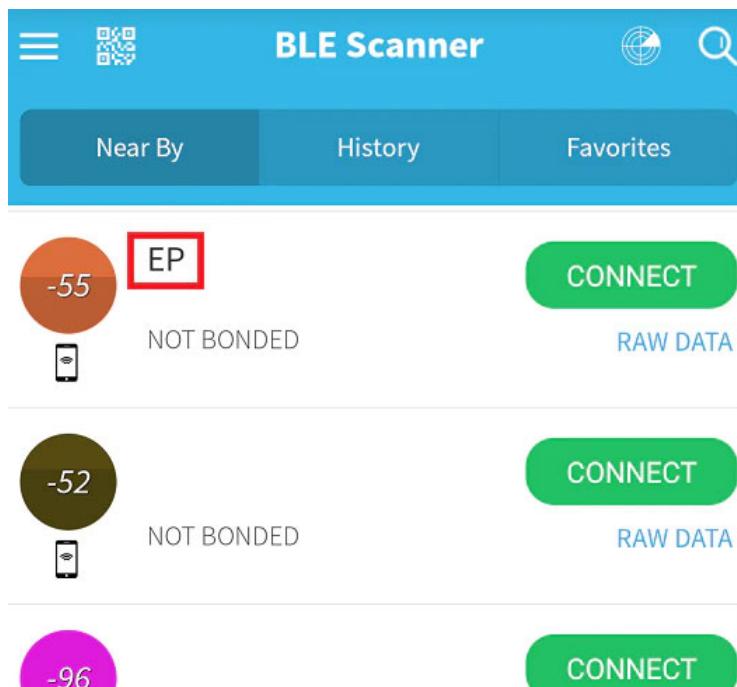
・アドバタイズ設定の変更

上図③「W」をクリックし、下図のように変更してOKを押します



・変更の確認

トップに戻ってデバイス名が「EP」に変わっていれば成功です



環境センサの更新間隔を短くする

センサ値の更新間隔はデフォルトで300秒（5分）となっています。
これでは長すぎる！という方が多いと思うので、短く変更します。

- ・ "0C4C3011～"が更新間隔設定なので、探す

前操作に引き続いで、BLE Scannerのデバイス「EP」にConnectし、
"0C4C3011～"を探します

PRIMARY SERVICE

CUSTOM SERVICE

- ▽ 0C4C3000-7700-46F4-AA96-D5E974E32A54
PRIMARY SERVICE

CUSTOM SERVICE

- ▽ 0C4C3010-7700-46F4-AA96-D5E974E32A54
PRIMARY SERVICE

CUSTOM SERVICE

- ▽ 0C4C3010-7700-46F4-AA96-D5E974E32A54

CUSTOM SERVICE

- ^ 0C4C3010-7700-46F4-AA96-D5E974E32A54
PRIMARY SERVICE

CUSTOM CHARACTERISTIC

UUID: 0C4C3011-7700-46F4-AA96-D5E974E32A54

Properties: READ,WRITE

Value:,

Hex: 0x2C01 **② 内容確認**

Write Type: WRITE REQUEST



UUID: 0C4C3012-7700-46F4-AA96-D5E974E32A54

・上図②の更新間隔設定の見方

秒単位の更新間隔を16進数に変換したものとなります。

ただし、上のバイトと下のバイトが逆転する初心者泣かせ仕様なので、
注意してください。

今回は下図のように10秒間に変更するため、"0A00"を入力します

	設定したい値	本来の16進数	実際の設定値
Before	300秒	012C	2C01
After	10秒	000A	0A00

・更新間隔の変更確認

「R」を押してHexの内容が更新されていれば成功です

CUSTOM SERVICE

0C4C3010-7700-46F4-AA96-D5E974E32A54

PRIMARY SERVICE

①

CUSTOM CHARACTERISTIC



UUID: 0C4C3011-7700-46F4-AA96-D5E974E32A54

Properties: READ,WRITE

Value:

Hex: 0x0A00

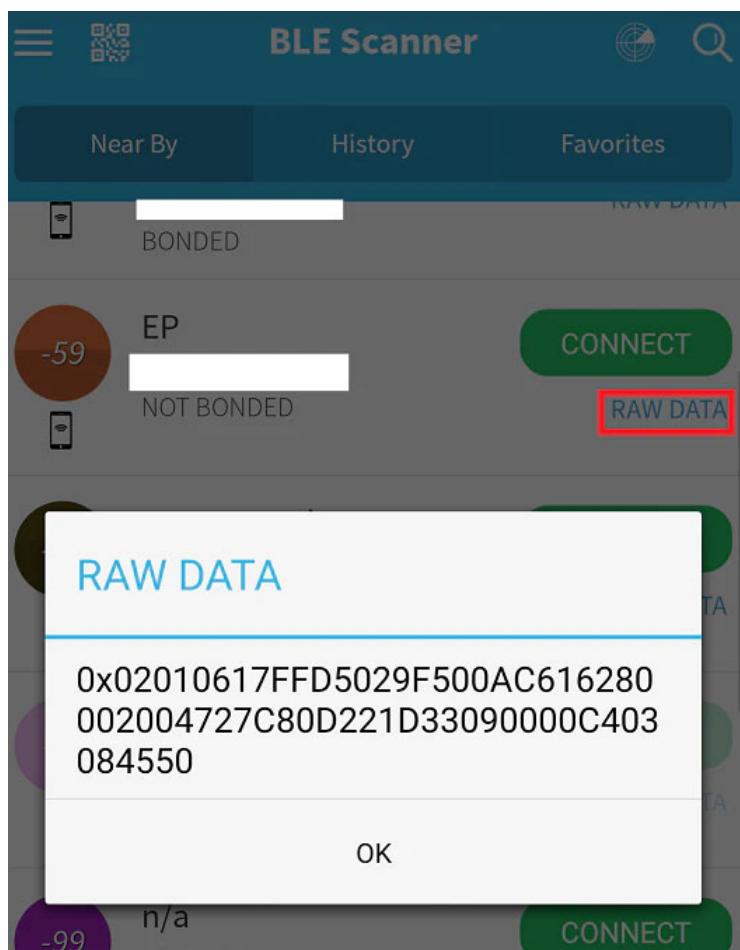
②

Write Type: WRITE REQUEST

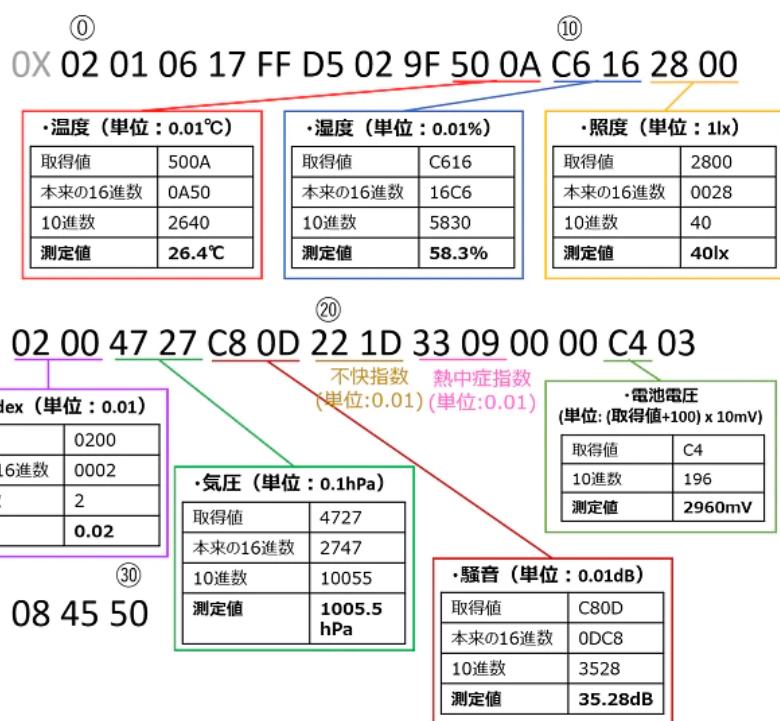
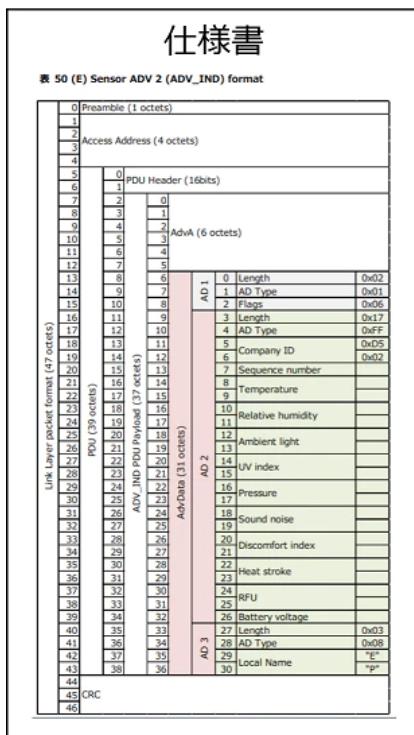
CUSTOM CHARACTERISTIC



なお、トップ画面で「RAW DATA」をクリックすると、センサの測定値を見る事ができます。



測定値の仕様は下図のようになっています



※例のごとくバイトの上下が逆転するのでご注意ください

③環境センサの測定値をPythonで取得

②でセンサの測定値が取得できることが分かったので、
Pythonで処理の汎用性をアップさせます。

センサ測定値取得用クラスを作成

bluepyでセンサ測定値を取得するクラスを作成します。

①と同様に、スキヤン時の処理をデリゲートとして渡すため、
下記のようなコードを作成します。

取得したセンサ測定値は変数「sensorValue」にセンサ種類ごとdictとして保持されます。

```
omron_env_broadcast.py

from bluepy import btle
import time
import struct

class ScanDelegate(btle.DefaultDelegate):
    # コンストラクタ
    def __init__(self):
        btle.DefaultDelegate.__init__(self)
        # センサデータ保持用変数
        self.sensorValue = None

    # スキヤンハンドラー
    def handleDiscovery(self, dev, isNewDev, isNewData):
        # 新しいデバイスが見つかったら
        if isNewDev or isNewData:
```

```

# アドバタイズデータを取り出し
for (adtype, desc, value) in dev.getScanData():
    #環境センサのとき、データ取り出しを実行
    if desc == 'Manufacturer' and value[0:4] == 'd502':
        #センサの種類（EP or IM）を取り出し
        sensorType = dev.scanData[dev.SHORT_LOCAL_NAME].decode(encoding='utf-8')
        #EPのときのセンサデータ取り出し
        if sensorType == 'EP':
            self.decodeSensorData_EP(value)
        #IMのときのセンサデータ取り出し
        if sensorType == 'IM':
            self.decodeSensorData_IM(value)

# センサデータを取り出してdict形式に変換（EPモード時）
def decodeSensorData_EP(self, valueStr):
    #文字列からセンサデータ(6文字目以降)のみ取り出し、バイナリに変換
    valueBinary = bytes.fromhex(valueStr[6:])
    #バイナリ形式のセンサデータを整数型tupleに変換
    (temp, humid, light, uv, press, noise, discomf, wbgt, rfu, batt) = struct.unpack('<hhhhhhhhB', valueBinary)
    #単位変換した上でdict型に格納
    self.sensorValue = {
        'SensorType': 'EP',
        'Temperature': temp / 100,
        'Humidity': humid / 100,
        'Light': light,
        'UV': uv / 100,
        'Pressure': press / 10,
        'Noise': noise / 100,
        'Discomfort': discomf / 100,
        'WBGT': wbgt / 100,
        'BatteryVoltage': (batt + 100) / 100
    }

# センサデータを取り出してdict形式に変換（IMモード時）
def decodeSensorData_IM(self, valueStr):
    #文字列からセンサデータ(6文字目以降)のみ取り出し、バイナリに変換
    valueBinary = bytes.fromhex(valueStr[6:])
    #バイナリ形式のセンサデータを整数型tupleに変換
    (temp, humid, light, uv, press, noise, accelX, accelY, accelZ, batt) = struct.unpack('<hhhhhhhhB', valueBinary)
    #単位変換した上でdict型に格納
    self.sensorValue = {
        'SensorType': 'IM',
        'Temperature': temp / 100,
        'Humidity': humid / 100,
        'Light': light,
        'UV': uv / 100,
        'Pressure': press / 10,
        'Noise': noise / 100,
        'AccelerationX': accelX / 10,
        'AccelerationY': accelY / 10,
        'AccelerationZ': accelZ / 10,
        'BatteryVoltage': (batt + 100) / 100
    }
}

```

下記部分で、アドバタイズデータからセンサ測定値を取得しています。

```

#環境センサのとき、データ取り出しを実行
if desc == 'Manufacturer' and value[0:4] == 'd502':
    #センサの種類（EP or IM）を取り出し
    sensorType = dev.scanData[dev.SHORT_LOCAL_NAME].decode(encoding='utf-8')

```

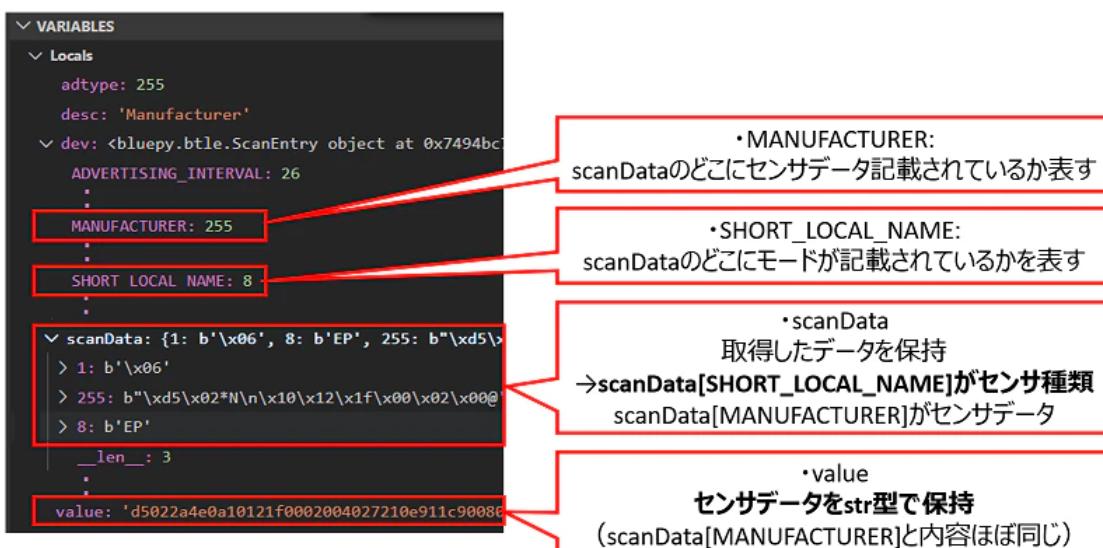
```
#EPのときのセンサデータ取り出し
if sensorType == 'EP':
    self.decodeSensorData_EP(value)
#IMのときのセンサデータ取り出し
if sensorType == 'IM':
    self.decodeSensorData_IM(value)
```

上記処理は、下記に基づいて作成しました。

- ・環境センサは、「desc='Manufacturer' & valueの0～4文字目が'd502'」で特定可能
- ・センサのモードおよび測定値は下記【参考】のような形で保持されている
- ・取得した測定値はバイナリデータで、かつモードがEPかIMかで形式が異なるため、利用しやすいよう数値Dict形式に変換するメソッド"decodeSensorData"をモードごとに作成

【参考】BroadCastモード時のアドバタイズデータをVSCodeでデバッグした結果

bluepyで取得したBroadCastモードのアドバタイズデータ



メインコードの作成

上記センサ値取得クラスを実行するための、メインコードを作成します。

- ②と同様Scannerのデリゲートにセンサ値取得クラスを設定し、
スキャンして実行します。

```
omron_env_toSpreadSheet.py

from bluepy import btle
from omron_env_broadcast import ScanDelegate

#omron_env_broadcast.pyのセンサ値取得デリゲートを、スキャン時実行に設定
scanner = btle.Scanner().withDelegate(ScanDelegate())
#スキャンしてセンサ値取得（タイムアウト5秒）
scanner.scan(5.0)
#試しに温度を表示
print(scanner.delegate.sensorValue['Temperature'])

#Googleスプレッドシートにアップロードする処理を④で記載
```

コンソールから実行してみます

```
python omron_env_toSpreadSheet.py
25.98
```

これで、Pythonでセンサ測定値を取得することができました。

取得した測定値はscanner.delegate.sensorValueに保持されています。

④PythonからGASのAPIを叩いてスプレッドシートにデータ書き込み

取得したデータをGoogleスプレッドシートに出力します。

※大半の人はスプレッドシート以外に出力したいと思うので、

その場合本章は飛ばし⑤に移ってください

(PandasでCSV出力したり、Ambientで可視化したりと選択肢は沢山あります)

GASスクリプトを作成

本作業は、PCで実施してください。

Googleスプレッドシートにアクセスし、

下記のようなスプレッドシートを作成します

なお、シート名はデバイス名と同じにしてください

	A	B	C	D	E	F	G	H	I	J
1	Date_Master	Date	Temperature	Humidity	Light	UV	Pressure	Noise	BatteryVoltage	Date_End
2										
3										
4										
5										
6										
7										
8										
9										
10										
11										
12										
13										

「ツール」→「スクリプトエディタ」を選択し、

下記のようなGASスクリプトを作成します

```
postSensorData.gs

var spreadsheetId = '*****' //スプレッドシートのIDを入れる

//Postされたデータを受け取り
function doPost(e){
  var data = [
    e.parameter.Date_Master, // マスター日時
    e.parameter.Date, // 測定日時
    e.parameter.Humidity, // 湿度
    e.parameter.Temperature, // 溫度
    e.parameter.Light, // 光照度
    e.parameter.UV, // UV指数
    e.parameter.Pressure, // 気圧
    e.parameter.Noise, // 音量
    e.parameter.BatteryVoltage // 電池電圧
  ];
  var sheet = SpreadsheetApp.getActiveSheet();
  sheet.appendRow(data);
}
```

```

e.parameter.Temperature, // 気温
e.parameter.Humidity, // 湿度
e.parameter.Light, // 照度
e.parameter.UV, // UV
e.parameter.Pressure, // 圧力
e.parameter.Noise, // 騒音
e.parameter.BatteryVoltage, // 電池電圧
new Date(), // アップロード終了時刻
];
// 取得したデータをログに記載
Logger.log(new Date());
// スプレッドシートへのデータ行書き込み
addData(e.parameter.DeviceName, data);
return ContentService.createTextOutput("ok");
}

// スプレッドシートへのデータ行書き込み
function addData(sheetName, data){
  var sheet = SpreadsheetApp.openById(spreadsheetId).getSheetByName(sheetName);
  sheet.appendRow(data);
}

```

※どうやらFunction名は「doPost」としないと、APIでPostできないようです

※スプレッドシートのIDは、URLが

```
https://docs.google.com/spreadsheets/d/AAAAAAA/edit
```

だとすると、"AAAAAAA"の部分が相当します

スクリプトをWeb APIとして公開

「公開」→「ウェブ アプリケーションとして導入」



下記のような画面が出てくるので、

- Project version → API化したい版（最初は「New」を選択すればOK）
 - Execute the app as → Me（自分）
 - Who has access to the app → Anyone, even anonymous（全員(匿名ユーザを含む)）
- を選択して更新（Deploy）

Project version:

1 ▾

Execute the app as:

Me ([REDACTED]) ▾

You need to authorize the script before distributing the URL.

Who has access to the app:

Anyone, even anonymous ▾

更新 **キャンセル** **ヘルプ**

※Google認証が出てきた場合、指示に従って認証

「このアプリは確認されていません」が出たら、下図のように「詳細」からリンクをたどればクリアできる



このアプリは確認されていません

このアプリは、Googleによる確認が済んでいません。よく知っている信頼できるデベロッパーの場合に限り続行してください。

詳細を非表示 ① **安全なページに戻る**

Googleではまだこのアプリを確認していないため、アプリの信頼性を保証できません。未確認のアプリは、あなたの個人データを脅かす可能性があります。
[詳細](#)

postAPITest (安全ではないページ) に移動 ②

成功すると、下記の画面が出てきます。

APIのURLはあとで使うので、大事にメモしてください。

Deploy as web app



This project is now deployed as a web app.

Current web app URL:

[https://script.google.com/macros/s/\[REDACTED\]/exec](https://script.google.com/macros/s/[REDACTED]/exec)

Test web app for your [latest code](#).

OK

API外部実行テスト

※ここからはRaspberryPi側での作業となります

APIをcurlで叩いて適切なデータを送信し、外部からPOSTできるかテストします。

下記コマンドを実行します（curlが入っていない場合はインストールしてください）

```
curl -L [APIのURL] -F 'sheetName=[シート名]' -F "Date_Master=4.1" -F "Date=4.2" -F "SensorType=4.2" -F "Temperature=4.4" -F "Humidi-
```

スプレッドシートに値が入力されれば成功です

The screenshot shows a Google Sheets document with a single sheet named 'シート1'. The columns are labeled A through K. Row 1 contains headers: Date_Master, Date, Temperature, Humidity, Light, UV, Pressure, Noise, BatteryVoltage, and Date_End. Row 2 contains values: 4.1, 4.2, 4.4, 4.4, 4.1, 4.2, 4.4, 4.4, 4.4, and 2020/05/06 13:35. The status bar at the bottom right indicates the document has been saved to Google Drive.

	A	B	C	D	E	F	G	H	I	J	K
1	Date_Master	Date	Temperature	Humidity	Light	UV	Pressure	Noise	BatteryVoltage	Date_End	
2	4.1	4.2	4.4	4.4	4.1	4.2	4.4	4.4	4.4	4.4	2020/05/06 13:35
3											
4											
5											
6											

※うまくいかないとき

私も何度か失敗しましたが、下記のようなGASスクリプトまわりが原因の事が多かったです。

- ・スクリプトのファンクション名が「doPost」になっていない
- ・スクリプトのAPI公開時に「doPost」関数を選択していなかった
- ・スクリプトのAPI公開時に最新バージョン（New）を選択していなかった

Requestsをインストール

Python側でPOSTを実行するため、Requestsライブラリをインストールします

```
pip install requests
```

Pythonコードにアップロード処理を追加

③で作成したコード「omron_env_toSpreadSheet.py」に、GASスクリプトのAPIを叩いてデータをアップロードする処理を追加します。

```
omron_env_toSpreadSheet.py

from bluepy import btle
from omron_env_broadcast import ScanDelegate
from datetime import datetime, timedelta
import requests

#現在時刻を取得
date = datetime.today()
#現在時刻を分単位で丸める
masterDate = date.replace(second=0, microsecond=0)
if date.second >= 30:
    masterDate += timedelta(minutes=1)

#omron_env_broadcast.pyのセンサ値取得デリゲートを、スキャン時実行に設定
```

```

scanner = btle.Scanner().withDelegate(ScanDelegate())
#スキャンしてセンサ値取得
scanner.scan(5.0)

#####Googleスプレッドシートにアップロードする処理#####
#センサ値がNoneでないときのみアップロード
if scanner.delegate.sensorValue is not None:
    #デバイス名
    deviceName = '*****'#<スプレッドシートのシート名と同じ名前にする
    #POSTするデータ
    data = {
        'DeviceName': deviceName,
        'Date_Master': str(masterDate),
        'Date': str(date),
        'SensorType': str(scanner.delegate.sensorValue['SensorType']),
        'Temperature': str(scanner.delegate.sensorValue['Temperature']),
        'Humidity': str(scanner.delegate.sensorValue['Humidity']),
        'Light': str(scanner.delegate.sensorValue['Light']),
        'UV': str(scanner.delegate.sensorValue['UV']),
        'Pressure': str(scanner.delegate.sensorValue['Pressure']),
        'Noise': str(scanner.delegate.sensorValue['Noise']),
        'BatteryVoltage': str(scanner.delegate.sensorValue['BatteryVoltage'])
    }
    #APIのURL
    url = 'https://script.google.com/macros/s/*****/exec'#+APIのURLを入れる
    #APIにデータをPOST
    response = requests.post(url, data=data)

```

※Date_Master列（masterDate変数）は複数台センサがあるときの結合キー用として作成したので、現時点では気にしないでください。

Pythonスクリプトを動かしてアップロードの確認

上記Pythonスクリプトを、下記コマンドで実行します

```
python omron_env_toSpreadSheet.py
```

下図のように、スプレッドシートにセンサの値が入力されれば成功です



The screenshot shows a Google Sheets spreadsheet with the following data:

Date_Master	Date	Temperature	Humidity	Light	UV	Pressure	Noise	BatteryVoltage	Date_End
2020-05-06 4:02	2020-05-06 4:02	25.72	43.21	22	0.01	1006.2	38.39	2.91	2020/05/06 4:02

⑤スクリプトの定期実行

上記の方法だと毎度スクリプトを実行する必要があり面倒なので、定期実行パッケージ「cron」を使用して自動化します。Windowsのタスクスケジューラのようなイメージです。

cronの有効化

デフォルトだと無効になっていることがあるので、[こちら](#)を参考に有効化します

・チェックポイント1：rsyslog.confの確認

/etc/rsyslog.confの「cron」がコメントアウトされていると動作しません。

自分の場合下記のようにコメントアウトされていたので、

```
#####
##### RULES #####
#####

#
# First some standard log files. Log by facility.
#
auth,authpriv.*          /var/log/auth.log
*.*;auth,authpriv.none    -/var/log/syslog
#cron.*
daemon.*                 -/var/log/daemon.log
```

下記のようにコメントアウトを外したのち、

```
cron.*                  /var/log/cron.log
```

下記コマンドでrsyslogを再起動します。

```
sudo /etc/init.d/rsyslog restart
```

・チェックポイント2：ログレベルの変更

/etc/default/cronで、cron実行時のログに記載する事項を指定します。

デフォルトでは下記のようにログ出力しないようなので、

```
# For quick reference, the currently available log levels are:
# 0  no logging (errors are logged regardless)
# 1  log start of jobs
# 2  log end of jobs
# 4  log jobs with exit status != 0
# 8  log the process identifier of child process (in all logs)
#
#EXTRA_OPTS=""
```

下記のようにEXTRA_OPTSのコメントアウトを外して

```
EXTRA_OPTS=' -L 15'
```

とします。1+2+4+8=15で全部出力しますという意味です。

下記コマンドでcronを再起動します。

```
sudo /etc/init.d/cron restart
```

/var/logにcron.logが生成されていれば成功です。
(cronの実行履歴を見たい場合もこちらを確認してください)

cronでスケジュール実行



25



23



...

・ crontabの編集

下記コマンドでcrontabを開きます

```
crontab -e
```

※「sudo crontab -e」は違うファイルが開かれるので、sudoを付けないで下さい

どのエディタで開くか聞かれた場合、好きなエディタを選択します（初心者はnano推奨）

```
# Edit this file to introduce tasks to be run by cron.  
#  
# Each task to run has to be defined through a single line  
# indicating with different fields when the task will be run  
: 色々続く
```

上のように色々コメントが書いてありますが、

一番最後に、実行したいタイミングとコマンドを記載します。

タイミングのフォーマットは[こちら](#)をご参照ください

今回は5分間隔で実行するため、下記内容を記載しました。

```
*/5 * * * * [Pythonのフルパス] [omron_env_toSpreadSheet.pyのフルパス] >/dev/null 2>&1
```

※[こちら](#)のように、Pythonはフルパス指定が必要です。フルパスの例は下記

- ・デフォルトのPython3 : /usr/bin/python3
- ・pyenv環境 : /home/[ユーザ名]/.pyenv/shims/python

※最後の「>/dev/null 2>&1」は、メールを送信しないための処理です。[こちら](#)参照

・ cronの起動

下記コマンドでcronを起動します

```
sudo /etc/init.d/cron start
```

上記でしばらく待ち、5分ごとにスプレッドシートが更新されれば完了です！

※うまくいかないときは、/var/log/syslogにあるcronのログを見て対処してください。

お疲れさまでした！

おわりに

センサの数を増やして、夢のIoTハウスを実現！なんてこともできそうです。
オムロンのセンサは多機能で質も高いですがポンポン買えるほど安くはないので、
もっと安いセンサでも試してみようと思います！

**Kenta Nakamura**

@c60evaporator

Python / Machine learning / Data science in manufacturing / IoT

フォロー



関連記事 Recommended by LOGLY

**SwitchBot温湿度計の値をRaspberryPiでログイン**

by c60evaporator

**家の中のセンサデータをRaspberryPiで取得しまくり、スーパーIoTハウスを実現**

by c60evaporator

**【IoT】スプレッドシートで、爆速で室内環境を可視化する**

by rueyjye

**AWS障害で我が家のIoTが逝ってしまったので、稼働管理システムを作った**

by c60evaporator

**豪華商品をプレゼント！Qiita Azure記事投稿キャンペーン**

PR 日本マイクロソフト

**“無料”セキュリティ対策「Windows Defender」を正しく学び直す**

PR アイティメディア

この記事は以下の記事からリンクされています

-  [窓の開閉を検知するセンサーを作る](#) からリンク 4 months ago
-  [洗濯物がでているかどうかを機械学習モデルで判別してgoogle spreadsheetへ出力する](#) からリンク 4 months ago
-  [1階と3階のraspberry piに接続している二酸化炭素濃度計のデータを1分おきにgoogle spread sheetへ記録](#) からリンク 6 months ago
-  [raspberry piで取得した温度・湿度をgoogle spread sheetへ記録](#) からリンク 6 months ago
-  [家の中のセンサデータをRaspberryPiで取得しまくり、スーパーIoTハウスを実現](#) からリンク 1 year ago

↑ 過去の4件を表示する

コメント



2021-09-18 09:07 ...

わかりやすい記事をありがとうございます。
こちらの記事を参考にさせていただき、ログインを楽しんでおります。

最近この1台raspberrypiにセンサーをもう一台増やして、2箇所のログ（西側窓と、玄関）を取りたいと思うようになりました。

センサー別にランダムで良いのでデータを取るような事はできるものでしょうか？

環境センサのとき、データ取り出しを実行 のあたりをうまく処理できればいいのでしょうか。。。？

いろいろ試してみたのですが、思うように行かず、なにか手がかりを頂けますと幸いです。



0



2021-09-28 12:33 ...

@taiszk さん

コメントありがとうございます。
まさに複数のセンサのデータを取得する場合の記事を作成しているので、以下をご参照頂ければと思います
<https://qiita.com/c60evaporator/items/283d0569eba58830f86e>



0



2021-10-03 20:21 ...

ありがとうございます。参考にさせていただきました。

BAG型センサーを複数台入れて計測したいと思いました。

技術的には可能なのでしょうか。。。？

色々と調べてみたいと思います。
ありがとうございました。



0



2021-10-03 20:27 ...

複数のBAG型センサーを使用する場合でも、MACアドレスが異なるのでいけるかと思います。上記リンク先の記事でいうと、DeviceList.csvに複数のBAG型センサーのMACアドレスをそれぞれ記載するイメージです。



1



御丁寧にありがとうございます。

また楽しみが増えました！



1

2021-10-03 22:16 ...



投稿する

編集 プレビュー

?



テキストを入力



0B / 100MB

投稿

記事投稿イベント開催中



マイクロソフト認定資格を取得する際の学習方法や経験談、おすすめ学習リソースなどを紹介しよう！

2021/11/11~2021/12/10

[イベント詳細を見る](#)

[すべて見る](#)

Qiita

How developers code is here.

© 2011-2021 Increments Inc.

ガイドとヘルプ^{*}

- About
- 利用規約
- プライバシーポリシー
- ガイドライン
- デザインガイドライン
- ご意見
- ヘルプ
- 広告掲載

コンテンツ

- リリースノート
- イベント
- アドベントカレンダー
- Qiita 表彰プログラム
- API
- Qiitadon (β)

SNS

- Qiita (キータ) 公式
- Qiita マイルストーン
- Qiita 人気の投稿
- Qiita (キータ) 公式

Qiita 関連サービス

- Qiita Team
- Qiita Jobs
- Qiita Zine
- Qiita 公式ショップ

運営

- 運営会社
- 採用情報
- Qiita Blog