検索

- チュートリアル
  - Ambientを使ってみる
  - Arduino ESP8266で温度・湿度を測定し、Ambientに送ってグラフ化する
  - チャネルとチャートのカスタマイズ
  - チャネルの公開
  - チャネルページへの写真の貼り込み
  - データーに位置情報を付加する
  - mbed LPC1768とAmbient
  - Raspberry Pi3とAmbient
  - Ambientのデーターをpythonで扱う
  - 稼働状態を可視化する
  - データ監視と通知
  - 複数台のIoT端末を同一プログラムで動かす
  - メーターと棒グラフ（現在値）
  - リストチャート、表示/非表示の制御、コメント
  - 状態表示
  - 箱ひげ図（Box plot チャート）
  - ダッシュボードに背景画像を設定する
- ライブラリー/リファレンス
  - Arduinoライブラリー
  - mbedライブラリー
  - 諸元/制限
  - Javascript / node.jsライブラリ ambient-lib
  - Pythonライブラリー
- サンプル
  - 温度・湿度・気圧・照度
  - M5Stack
  - ラズベリーパイ
  - 振動
  - 電流
  - 心拍
  - データー処理
  - IoTデバイス
  - ネットワーク
- 事例
  - 製造業：鋼材の焼き入れ
  - 3密、換気対策：Co2ロギングメーター
  - 農業：いちごと苗の生産
  - 研究・教育：環境情報の観測
  - スポーツ：ソーラーカーレースチーム TeamMAXSPEED
- 書籍
  - IoT開発スタートブック ── ESP32でクラウドにつなげる電子工作をはじめよう！

タグ

api Arduino BLE Bluetooth BME280 bulk_send dashboard ESP-WROOM-02 ESP32

オムロン環境センサーからBLE経由でデーターをクラウドに送る

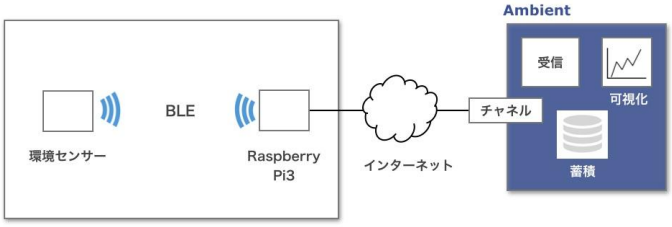
オムロンの環境センサー「2JCIE-BL01」で測定した温度、湿度などのデーターを、Bluetooth Low Energy(BLE)でRaspberry Piに送り、Raspberry Pi3からAmbientに送って可視化します。Raspberry Pi3のプログラムはPythonで記述しました。

オムロンの環境センサー「2JCIE-BL01」

オムロンの環境センサー「2JCIE-BL01」は温度、湿度、気圧、照度、UV、音の6種類のデーターを測定し、BLEで送信するセンサー端末です。ボタン電池(CR2032)1個で動作します。ウェザーニュース社の「WxBeacon2」も中身は同じもののようです。今回はこの「WxBeacon2」を使いました。

「2JCIE-BL01」のBLEインタフェースは「Communication Interface Manual」という資料で公開されています。

全体の構成



センサー端末から温度、湿度などのデーターをBluetooth Low Enrgy (BLE)で送ります。Raspberry Pi3をゲートウェイにしてBLEでデーターを受信し、Ambientに送信して可視化します。BLEではセンサー端末をペリフェラル、ペリフェラルにデーターを取りにいくものをセントラルといいます。今回の場合、Raspberry Pi3がセントラルになります。

Raspberry Pi3の準備

Raspberry Pi3はBluetooth Low Energyの通信モジュールが搭載されているので、追加するハードウェアはありません。Raspberry Pi3のOSは最新のRaspbian GNU/Linux 9.4 (stretch)を使いました。jessieでも動作すると思います。

プログラミング言語では、Pythonとnode.jsにBLEを扱うライブラリーがあります。今回はPythonを使うことにします。Raspberry Pi3 stretchにはPython2.7とPython3がインストールされていますが、pyenvを導入し、環境を切り替えて使います。pyenvの使い方は「RaspberryPiにpyenvを導入」などをご覧ください。Pythonのバージョンは3.6.5を使いました。

PythonでBLEを扱うライブラリーはいくつかあります。その中でドキュメントが一番しっかりしているbluepyを使いました。次のようにインストールします。pi$と書かれたものはRaspberry Pi3上のコマンド実行です。

```
pi$ sudo apt-get install libglib2.0-dev
pi$ pip install bluepy
```

pyenvを使わず直接python3を使う方は、pipの代わりにsudo pip3としてください。

Ambientのライブラリーもインストールし ます。

```
pi$ pip install git+https://github.com/AmbientDataInc/ambient-python-lib.git
```

環境センサー「2JCIE-BL01」とRaspberry Pi3の動作確認

ESP8266 FFT GPS HDC1000 IoTセミナー javascript LoRa LPC1768 M5Stack m5stickc matplotlib mbed mbed祭り microbit micropython Node-RED node.js Obniz pandas python Raspberry Pi raspberry pi3 SensorTag Simple IoT Board Wi-Fi セミナー 位置情報 加速度センサー 心拍センサー 心拍モニター 振動 消費電流 環境センサー 環境モニター 見える化 電流センサー

Raspberry Pi3のBluetoothのバージョンは5.43です。

```
pi$ bluetoothd -v
5.43
```

「2JCIE-BL01」に電池を入れ、hcitoolコマンドでBLEのスキャンをしてみます。

```
pi$ sudo hcitool lescan
LE Scan ...
EC:B3:46:BA:78:75 Env
```

スキャンで見つかったBLE端末が複数表示されます。その中で名前が「Env」のものが「2JCIE-BL01」です。名前の左(例ではEC:B3:46:BA:78:75)が「2JCIE-BL01」のアドレスです。

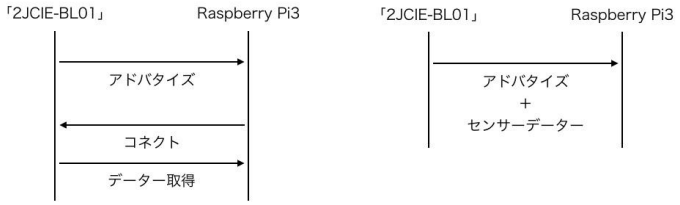
次にgatttoolコマンドで「2JCIE-BL01」にコネクトし、サービスを調べます。hcitoolで見つけた「2JCIE-BL01」のアドレスを-bオプションで指定します。最後の「-I」は大文字の「i」です。

```
pi$ gatttool -b EC:B3:46:BA:78:75 -t random -I
[EC:B3:46:BA:78:75][LE]> connect
Attempting to connect to EC:B3:46:BA:78:75
Connection successful
[EC:B3:46:BA:78:75][LE]> primary
attr handle: 0x0001, end grp handle: 0x0007 uuid: 00001800-0000-1000-8000-00805f9b34fb
attr handle: 0x0008, end grp handle: 0x000b uuid: 00001801-0000-1000-8000-00805f9b34fb
attr handle: 0x000c, end grp handle: 0x0016 uuid: 0000180a-0000-1000-8000-00805f9b34fb
attr handle: 0x0017, end grp handle: 0x0025 uuid: 0c4c3000-7700-46f4-aa96-d5e974e32a54
attr handle: 0x0026, end grp handle: 0x003a uuid: 0c4c3010-7700-46f4-aa96-d5e974e32a54
attr handle: 0x003b, end grp handle: 0x0043 uuid: 0c4c3030-7700-46f4-aa96-d5e974e32a54
attr handle: 0x0044, end grp handle: 0xffff uuid: 0c4c3040-7700-46f4-aa96-d5e974e32a54
```

「connect」コマンドを入力し、しばらくして「Connection successful」が表示されればコネクトに成功しています。「primary」コマンドを入力すると「2JCIE-BL01」で定義されているサービスが確認できます。1行目の「uuid: 00001800-・・・」の5文字目から8文字目までの1800が「Generic Access Service」というサービスUUIDで、端末の名前などが取得できます。詳細は「2JCIE-BL01」のマニュアルをご覧ください。

プログラム1(コネクトモード)

「2JCIE-BL01」はアドバタイズして、セントラルに見つけてもらい、セントラルからコネクトされてデーターのやり取りやモード設定などをおこなうモード(コネクトモードと呼ぶことにします)と、アドバタイズデーターにセンサー値を載せて送ってしまうモード(ブロードキャストモードと呼びます)があります。



コネクトモード　　ブロードキャストモード

デフォルトではコネクトモードになっているので、まずコネクトモードの「2JCIE-BL01」からセンサーデーターを取得してAmbientに送信するRaspberry Pi3のプログラムを作ります。PythonのBLEライブラリーbluepyのクラス定義の資料を参照しながらお読みください。

スキャンをおこなうにはScannerクラスのオブジェクトを作り、scanメソッドを呼びます。このとき、withDelegateでスキャンハンドラーを指定しておくと、デバイスを見つけ次第ハンドラーが呼ばれます。スキャンとスキャンハンドラーは次のようなプログラムになります。

```python
class ScanDelegate(DefaultDelegate):
    def __init__(self):
        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 = Scanner().withDelegate(ScanDelegate())
while True:
    scanner.scan(10.0) # スキャンする。デバイスを見つけた後の処理はScanDelegateに任せる
```

ble_scan.py hosted with ❤ by GitHub　　view raw

スキャンハンドラーに渡されるdevはScanEntryクラスのオブジェクトで、getScanDataでアドバタイズデーターを取り出せます。「2JCIE-BL01」はアドバタイズデーターとしてデータータイプが'Short Local Name'で、値が'Env'という情報を送るので、

次のようにすると「2JCIE-BL01」を見つけられます。見つけたあとは「2JCIE-BL01」にコネクトしてセンサーデーターを取得するので、そのためにスレッドを起動しています。

```
    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
            for (adtype, desc, value) in dev.getScanData():  # アドバタイズデーターを取り出し
                if desc == 'Short Local Name' and value == 'Env':  # 「2JCIE-BL01」を見つけ
                    if dev.addr in devs.keys():
                        return
                    MSG('New %s %s' % (value, dev.addr))
                    devThread = EnvSensor(dev)  # EnvSensorオブジェクトを生成し
                    devs[dev.addr] = devThread
                    devThread.start()  # 起動する
```

ble_scan_handler.py hosted with ❤ by GitHub view raw

BLEデバイスと通信するためにbluepyではPeripheralというクラスが用意されています。今回は5分ごとに「2JCIE-BL01」にコネクトしてセンサーデーターを取得し、Ambientに送るスレッドを作るので、PeripheralクラスとThreadクラスを多重継承したEnvSensorというクラスを作りました。コンストラクターでは親クラスのコンストラクターを呼び、Ambientの初期化もおこないます。

```
class EnvSensor(Thread, Peripheral):
    def __init__(self, dev):
        Peripheral.__init__(self)
        Thread.__init__(self)
        self.setDaemon(True)
        self.dev = dev
        self.am = ambient.Ambient(channelID, writeKey)
```

ble_EnvSensor_init.py hosted with ❤ by GitHub view raw

スレッドの処理はrunメソッドに記述します。中身はデバイスにコネクトして、
センサーデーターを取得し、Ambientに送信して300秒(5分)待つという処理の繰り返しです。

```
    def run(self):
        while True:
            self.connect(self.dev)  # デバイスにコネクトする
            latestDataRow = self.getCharacteristics(uuid=_OMRON_UUID(0x3001))[0]
            dataRow = latestDataRow.read()
            send2ambient(self.am, dataRow)
            time.sleep(300.0)
```

ble_EnvSensor_run.py hosted with ❤ by GitHub view raw

「2JCIE-BL01」はSensor Serviceの中に最新データー(Latest data)というCharacteristicがあり、これをreadすると温度、湿度などの最新データーが取得できます。デフォルトの測定間隔は300秒(5分)です。getCharacteristicsでLatest dataのUUID(0x3001)を指定してCharacteristicを取得し、readして、最新のセンサーデーターを取得しています。データーは温度、湿度などが19バイトにまとめららたbytesタイプです。それをそのままAmbientに送る関数send2ambientに渡しています。

```
def send2ambient(am, dataRow):
    (seq, temp, humid, light, uv, press, noise, discom, heat, batt) = struct.unpack('<BhhhhhhhhH', data
    am.send({'d1': temp / 100, 'd2': humid / 100, 'd3': press / 10, 'd4': batt / 1000, 'd5': light, 'd
```

ble_send2ambient.py hosted with ❤ by GitHub view raw

send2ambientで19バイトにまとめられたbytesタイプのデーターをstruct.unpackを使って温度、湿度など個々のデーターに分解し、Ambientに送信しています。

scanやconnect、getCharacteristics、readなどのメソッドの中でBLEデバイスと通信します。通信に失敗した時などには例外が発生するので、実際のプログラムには例外処理が加えてあります。プログラム全体はGithubに公開しました。

- EnvSensorBleGw

Raspberry Pi3のBLEモジュールにアクセスするにはroot権限が必要なので、プログラムは次のように起動します。

```
 pi$ sudo python env2ambientCS.py
New Env ec:b3:46:ba:78:75 ←スキャンして「2JCIE-BL01」を見つけた
connected to ec:b3:46:ba:78:75 ←「2JCIE-BL01」にコネクト
11 25.94 57.2 263 0.02 1010.7 37.71 73.82 23.0 2.956 ←最新のセンサーデーターを取得
sent to Ambient (ret = 200) ←Ambientに送信
```

Raspberry Pi3からログアウトしてもプログラムが終了しないようにするには、次のように起動します。

```
 pi$ sudo nohup python env2ambientCS.py < /dev/null &
```

Ambientのチャネルページを見ると、送られたデーターを確認できます。

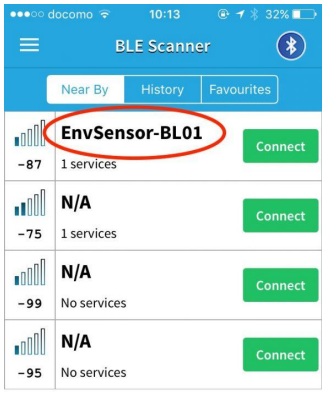
プログラム2(ブロードキャストモード)

「2JCIE-BL01」のモード変更

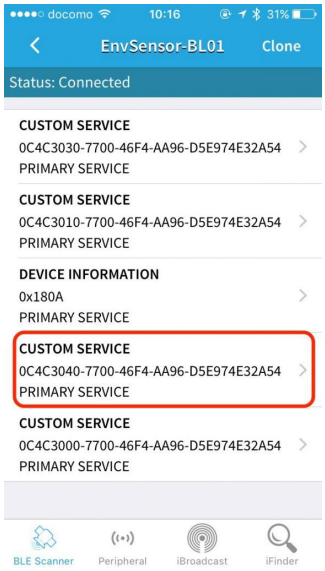
「2JCIE-BL01」にはアドバタイズデーターにセンサー値を載せて送るブロードキャストモードがあります。更にブロードキャストモードの動作として一定間隔(デフォルトで1.285秒間隔)で発信するモードと10秒発信して50秒休むというように間欠動作するモードの二つがあります。発信と休止の時間幅も変更できます。休止中は通信モジュールをオフにしているようで、コネクトもリード/ライトもできなくなり、消費電力を抑えています。「2JCIE-BL01」を間欠動作のブロードキャストモードに設定し、送られたデーターを取得してAmbientに送信するプログラムを作ります。

アドバタイズの動作設定はParameter Service(UUID: 0x3040)のADV settingというCharacteristic(UUID: 0x3042)でおこないます。プログラムを作ってもいいのですが、スマホアプリでやってみます。アプリは「BLE Scanner」を使います。

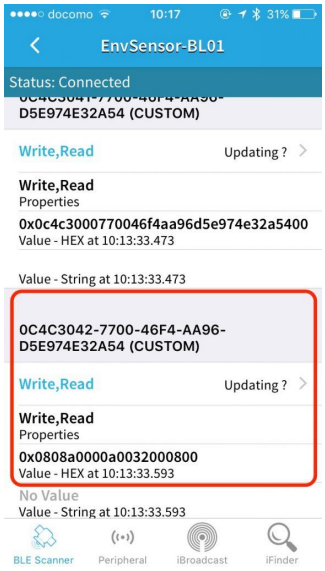
「BLE Scanner」を立ち上げると、スキャンして見つかったBLEデバイスが表示されます。



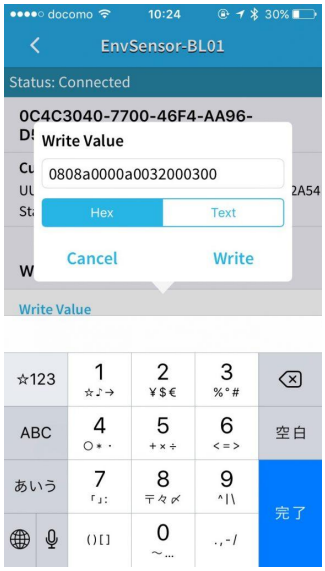
「EnvSensor-BL01」と表示されたものが環境センサー「2JCIE-BL01」です。それをタップするとアドバタイズデーターとサービスが表示されます。サービスの中に「0C4C3040-7700-・・・」というUUIDのものがあり、これがParameter Serviceです。



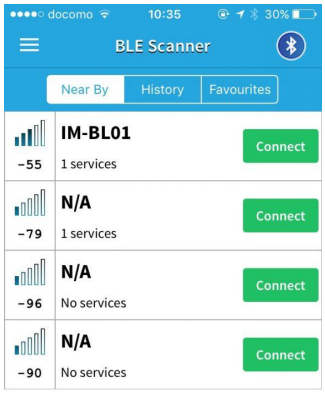
これをタップすると、Characteristicが表示されます。「0C4C3042-7700-・・・」というUUIDのものがADV settingです。



値が「0x0808a0000a0032000800」と表示されています。9バイト目がビーコンモードで「08」はEventBeaconを表します。これを「03」のLimited Broadcaster 1に変更します。Limited Broadcaster 1が間欠動作モードです。このCharacteristicをタップして、値を0808a0000a0032000300にして書き込みます。



値を書き込んだら「2JCIE-BL01」の電池を入れ直してリセットすると、新しいモードで動き出します。もう一度「BLE Scanner」を立ち上げると、デバイスの名前がIM-BL01に変わっていることが確認できます。



Raspberry Pi3側のプログラム

ブロードキャストモードの時のRaspberry Pi3側のプログラムもまずスキャンをします。スキャンハンドラーを指定してスキャンを起動するところまではコネクトモードの時のプログラムと同じです。

スキャンハンドラーに渡されるdevのgetScanDataメソッドを呼び、アドバタイズデーターを取り出します。ブロードキャストモードの時「2JCIE-BL01」はアドバタイズデーターの中にデータータイプが'Manufacturer'で、その時の値の先頭の文字列が'd502'という情報を送るので、次のようにすると「2JCIE-BL01」のアドバタイズデーターを見つけられます。

```
    def handleDiscovery(self, dev, isNewDev, isNewData):
            if isNewDev or isNewData:
                for (adtype, desc, value) in dev.getScanData():
                    if desc == 'Manufacturer' and value[0:4] == 'd502':
                        if value[4:6] != self.lastseq:
                            self.lastseq = value[4:6]
                            send2ambient(value[6:])
```

ble_broadcast_handler.py hosted with ❤ by GitHub view raw

値の5文字目、6文字目(value[4:6])はシーケンス番号で、測定をおこなうごとに値が一つ増やされます。デフォルトの測定間隔は300秒(5分)なので、5分に1回値が増えます。シーケンス番号が更新されたらデーターをAmbientに送信します。

Ambientに送信する関数send2ambientもコネクトモードの時とほぼ同じです。コネクトモードの時はCharacteristicをreadした値はbytesタイプでしたが、スキャンで得られるデーターは文字列なので、bytes.fromhexでbytesタイプに変換し、温度、湿度などのデーターに分解しています。

```
def send2ambient(dataRow):
    (temp, humid, light, uv, press, noise, accelX, accelY, accelZ, batt) = struct.unpack('<hhhhhhhhhB'
    am.send({'d1': temp / 100, 'd2': humid / 100, 'd3': press / 10, 'd4': (batt + 100) / 100, 'd5': li
```

ble_broadcast_send2ambient.py hosted with ❤ by GitHub view raw

プログラムは次のように起動します。

```
pi$ sudo python env2ambientBS.py
```

プログラム全体はGithubに公開しました。

- EnvSensorBleGw