

# 照度データWeb表示システム 構築手順書

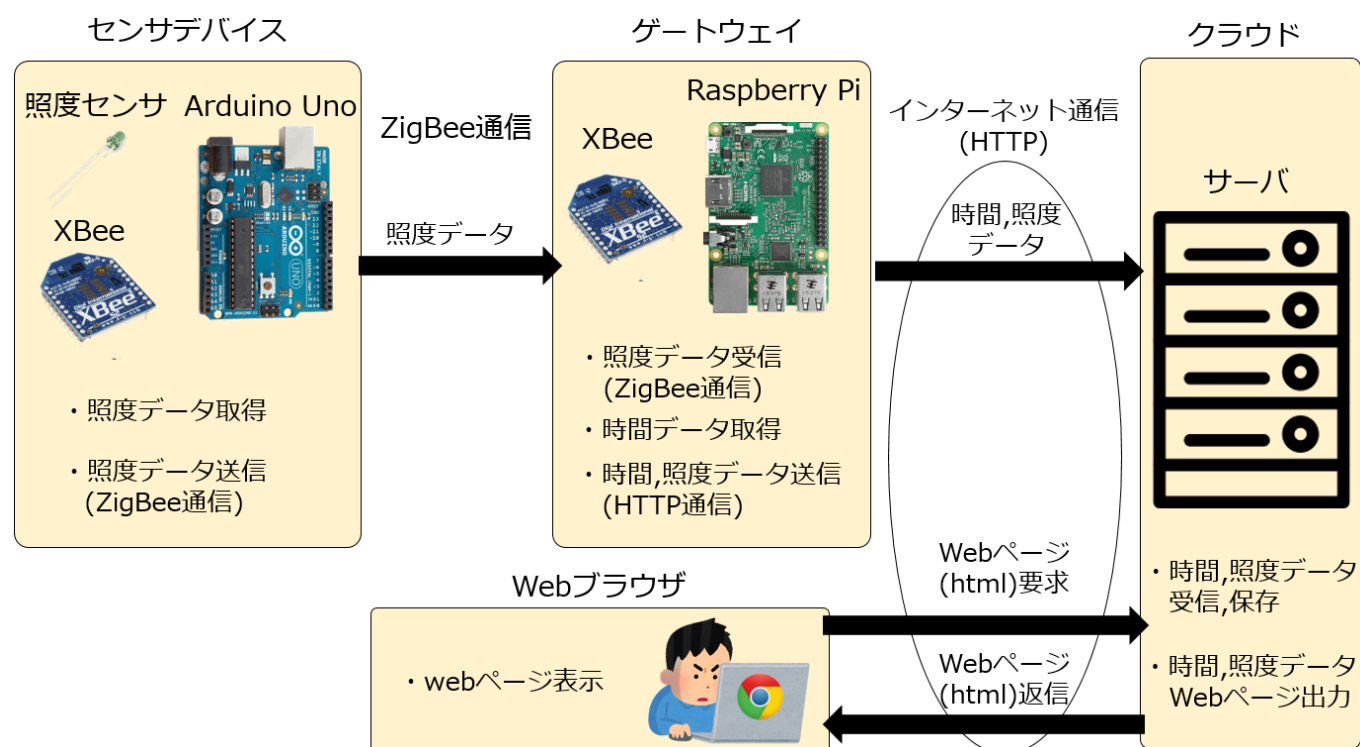
# 本手順書の目的

本手順書は、照度データWeb表示システムというIoTシステムを構築することにより、IoTシステムの構築方法を理解することを目的としている。

## 照度データWeb表示システム概要

### システム構成

照度データWeb表示システムの構成図を以下に記す。



### 機能

また、照度データWeb表示システムは以下の機能を持つ。

- 照度データ保存機能
  - センサデバイスの照度センサを用いて照度データを取一定間隔で取得、ZigBee通信でゲートウェイに送信。
  - ゲートウェイでセンサデバイスからの照度データを受信、現在の時間情報を付加してHTTP通信で照度データをクラウドに送信。
  - クラウドで時間と照度データを受信、ファイルに保存。
- 温度データ表示機能
  - ブラウザからサーバにHTTP通信でhtmlを要求。
  - html上のJavaScriptが実行、時刻と照度データを表示。
  - ブラウザで更新ボタンをクリックすると、時刻とp照度データを更新

# 事前準備

---

このIoTシステムを構築するにあたり,以下のソフトウェアのインストールなどが必要である.

- GitのインストールおよびGitHubへの登録
- cURLのインストール
- Pythonのインストール
- Visual Studio Codeのインストール
- eduroamの登録
- Arduino IDEのインストール
- X-CTUのインストール
- ブラウザのインストール

以下,それぞれの概要とインストール方法などを記述する.

## GitのインストールおよびGitHubへの登録

---

### Gitについて

Gitとはプログラムのソースコードなどを管理する分散型のバージョン管理ツールである. バージョン管理ツールとは,編集されるファイルの変更履歴を管理するためのシステムである. また,Gitのクラウドサービスを提供する[Github](#)や[BitBucket](#)が存在する.

### Gitの使い方

Gitを用いたソフトウェアのバージョン管理の方法を簡単に説明する.

まず,Gitのインストールを行う. 以下のurlから,自分のPCのOSとあっているものを選択,インストールする.

<https://git-scm.com/downloads>

インストールが終了したら,ターミナルやコマンドプロンプトなどで自分のソースコードを編集しているディレクトリのルートディレクトリに移動する. そのディレクトリで以下のコマンドを入力する.

```
git init
```

このコマンドを入力すると, .gitディレクトリが作成され,このディレクトリが作成されたディレクトリ下のファイルのバージョン管理を行うことができる.

次に,Gitのコミット(変更履歴の登録)を説明する. Gitのコミット方法としては,プロジェクト内のファイルの追加・変更後, 以下のコマンドを入力し.

```
git add .
```

さらに,以下のコマンドを入力することで gitに変更履歴が記録される

```
git commit -m "コミットメッセージ"
```

コミットメッセージには変更した内容を記述する.

例えば,

```
git commit -m "ユーザ認証機能を追加"  
git commit -m "問題になっていたバグXXを修正"
```

といった風に記述を行う.

## まとめ

Gitでのプロジェクト管理方法は

1. git init (1回だけ)
2. プロジェクト内のファイル内容の追加・変更を行う.
3. git add .
4. git commit -m "コミットメッセージ"
5. 2 ~ 4 を繰り返す

となる

## GitHubについて

次に,GitHubの説明をする. GitHubとはgitのプロジェクトをクラウド上に保存できるサービスであり, 多種多様なOSSの開発がGitHub上で行われている. 本手順書ではこのサービスを利用してプログラムの管理とデプロイを行う.

### GitHubへの登録

以下のurlからGitHubのサイトへいき,Sing Upをクリック

<https://github.com/>

GitHubにアカウントを登録する.

また,GitHubには二種類のリポジトリがあり,それぞれの特徴を記す.

- パブリックリポジトリ  
自分以外も閲覧可能なリポジトリ  
自分以外にプロジェクトを公開したい時に使う.
- プライベートリポジトリ  
自分と自分が認可した人以外は閲覧が不可能なリポジトリ  
パブリックリポジトリでは困る時,例えばパスワードなどをソースコード内に含めるときに使用する.

# cURLのインストール

---

## cURLとは

cURLとはさまざまなプロトコルを用いてデータを転送するライブラリとコマンドラインツールを提供するプロジェクトであり、  
本手順書ではHTTP通信のテストのために使用する。

## インストール方法

MacOSやLinuxでは標準でインストールされている。Windowsの場合にはWindows10であれば、  
Windows 10 Ver.1803（RS4）のプレビュー版、Build 17063からならば、標準でインストールされているので  
インストール不要である。

インストールされているかどうかの確認方法としては、

- MacOS,Linuxの場合,ターミナルで

```
which curl
```

と入力し,パスが表示されればインストールされている。

- Windowsの場合,コマンドプロンプトか,パワーシェルで

```
Where curl
```

と入力し,パスが表示されればインストールされている。

インストールされていない場合には、ここ <https://curl.haxx.se/download.html> からダウンロードし,インストールする。

## ブラウザのインストール

---

JavaScriptが正常に動作しない可能背があるので、  
Internet Explorer**以外**をインストールする。  
インストールされている場合は,最新版にアップデートする。

# Pythonのインストール

---

## Pythonとは

Pythonとはスクリプト言語という種類の言語であり,C言語やJavaで必要であったコンパイルが不要である. また,動的型付けという実行時に型を決定方式を利用しているため,変数や関数の型宣言が不要である(型が存在しないという訳ではない). 現在,PythonにはPython2とPython3が存在するが,Python3が最新版のPythonであるためPython3の方を利用する.

## インストール方法

公式サイト(<https://www.python.org/downloads/>)から自分のPCに適したpython3をダウンロード.  
ダウンロードするバージョンは最新版(執筆時(2019/2/17)の最新版はPython3.7.2)をダウンロードする.

## pip3について

pip3というpython3のパッケージマネージャ(インストールやアンインストールなどのソフトウェアの管理を行う)ものである. pip3ならばpython3モジュールの管理を行う)があり,これを利用してIoTシステム構築に必要なpython3モジュールをインストールする.

## Visual Studio Codeのインストール

---

Visual Studio Codeはソースコードを記述するためのエディタである.  
Emacsやviなどのエディタも存在するが,これらは扱いにくいので,  
Visual Studio Codeを使用する. 以下のurlからインストールする.  
<https://code.visualstudio.com/>

# eduroamの登録

---

eduroamとは,複数の大学が共同で提供しているWi-Fi通信サービスである. 本手順書では,[さくらインターネット](#)という外部のVPSを利用するが, SRASではネットワークに制限がかかっているためさくらインターネットに接続ができない. そこで,本手順書では[eduroam](#)というネットワークに制限がないWi-FWサービスを利用することにより,さくらインターネットに接続する.

## Arduino IDEのインストール

---

センサデバイスで使用するArduino Unoにプログラムを書き込むために, 必要なソフトウェアであるArduino IDEを以下のurlからダウンロードし,インストールする.

<https://www.arduino.cc/en/main/software>

## X-CTUのインストール

---

センサデバイスとゲートウェイの間の通信にZigBee通信を用い, ZigBee通信を行うための通信機器として,XBeeを利用する.

このXBeeの設定をするためのソフトウェアがX-CTUである.以下のurlからダウンロードし,インストールする.

<https://www.digi.com/products/iot-platform/xctu#resources>

# 構築手順

---

本手順書では,以下の手順でシステムを構築していく.

1. サーバの作成
2. センサデバイスの作成
3. ゲートウェイの作成

## 1. サーバプログラムの作成

---

### 1.1 サーバプログラムプロジェクト初期化

サーバサイドのプログラムプロジェクトであるserverディレクトリを作成.  
serverディレクトリを作成後,git init コマンドを入力しプロジェクトを初期化する.  
以下にこの手順のコマンドの入力を記す.

```
mkdir server  
cd server/  
git init
```

入力例

```
yamazaki@TsuyoshinoMacBook-puro:~$ mkdir server  
yamazaki@TsuyoshinoMacBook-puro:~$ cd server/  
yamazaki@TsuyoshinoMacBook-puro:~/server$ git init  
Initialized empty Git repository in /Users/yamazaki/server/.git/  
yamazaki@TsuyoshinoMacBook-puro:~/server$
```

### 1.2 Flaskインストール

次にサーバプログラムで使用するFlaskというフレームワークをインストールする.  
Flaskとは,pythonを用いたRESTful APIの開発を容易にするフレームワークである.

インストール方法

```
pip3 install flask
```

上記のコマンドを入力すればインストールできる.



## 1.3 サーバプログラム作成

次に,センサデータの受信とブラウザへのセンサデータの表示を行うプログラム,server.pyを作成する.  
server.pyの内容を以下に記す.

server.py

```
from flask import Flask, request, render_template
app = Flask(__name__)
file_path = "./sensor_data.csv"
my_port = 17123

@app.route('/', methods=['GET'])
def get_html():
    return render_template('./index.html')

@app.route('/lux', methods=['POST'])
def update_lux():
    time = request.form["time"]
    lux = request.form["lux"]
    try:
        f = open(file_path, 'w')
        f.write(time + "," + lux)
        return "succeeded to write"
    except Exception as e:
        print(e)
        return "failed to write"
    finally:
        f.close()

@app.route('/lux', methods=['GET'])
def get_lux():
    try:
        f = open(file_path, 'r')
        for row in f:
            lux = row
        return lux
    except Exception as e:
        print(e)
        return e
    finally:
        f.close()

if __name__ == '__main__':
    app.run(debug=True, host='0.0.0.0', port=my_port)
```

また,templatesというディレクトリを作成し,その下に以下のindex.htmlを作成する.

index.html

```
<title>テストアプリ</title>
<html>
  <head>
    <script>
      const updateLux = async(url, element) => {
        const sensorData = await fetch(url)
                              .then(response => response.text())
        const target = document.getElementById(element)
        target.innerHTML = `<h1>${sensorData}</h1>`
      }
    </script>
  </head>
  <center>
    <body onLoad="updateLux('/lux', 'data')">
      <div id = "data">test</div>
      <input type="button" value="更新" onclick="updateLux('/lux', 'data')">
    </body>
  </center>
</html>
```

このプログラムは `python3 server.py` とコマンドを入力することで実行でき, 実行に成功すると以下の出力が表示される.

```
* Serving Flask app "server" (lazy loading)
* Environment: production
  WARNING: Do not use the development server in a production environment.
  Use a production WSGI server instead.
* Debug mode: on
* Running on http://0.0.0.0:17123/ (Press CTRL+C to quit)
* Restarting with stat
* Debugger is active!
* Debugger PIN: 216-426-961
```

また,この作業が終了した時のディレクトリ構成は以下のようになる.

```
server
├── server.py
└── templates
    └── index.html
```

# server.py解説

server.pyの解説を機能と部分ごとにしていく.

## プログラムの機能

このプログラムは以下の機能を持つ.

- ルートパスにGETメソッドでアクセスするとindex.htmlをリターン
- /luxパスにGETメソッドでアクセスすると,sensor\_data.csvのファイル内容(時間,照度値)をリターン
- /luxパスにPOSTメソッドと時間と照度値のパラメータを添付してアクセスすると,  
sensor\_data.csvにその内容を**上書き**で書き込む

それぞれについて,プログラムの部分ごとに説明していく

## 変数定義部分

```
from flask import Flask, request, render_template
app = Flask(__name__)
file_path = "./sensor_data.csv"
my_port = 17123
```

この部分では,以下の定義を行なっている.

- Flaskで使用するモジュールのインポート
- センサデータを保存するファイルパス(file\_path)の定義
- プログラムが使用する**ポート番号**(my\_port)の定義

また,my\_portの数字は0~65535の中から,0~1023**以外**の数字を使用する.

理由としては,これらの番号はウェルノウンポート番号と呼ばれるポート番号であり, 使用するには管理者権限が必要なためである. なので,my\_portの番号は自分の学籍番号ALXXXXXのXXXXXの部分を使用すると良い.

## htmlリターン部分

```
@app.route('/', methods=['GET'])
def get_html():
    return render_template('./index.html')
```

この部分では,ルートパスにHTTPのGETメソッドでアクセスされた場合に,index.htmlを返すコードを記述している.

このプログラムでは,ブラウザからlocalhost:17123(17123はmy\_portの番号)にアクセスすると,

1. サーバはindex.htmlをブラウザにリターン
2. ブラウザはindex.htmlの内容を解釈し表示
3. 最終的に以下のような画面がブラウザに出力される.

localhost:17123

2019/02/24 22:47:25,700

更新

カンマ区切りで左が時刻,右が照度値である.

という順番に処理される.

## センサデータ受信部分

```
@app.route('/lux', methods=['POST'])
def update_lux():
    try:
        time = request.form["time"]
        lux = request.form["lux"]
    except Exception as e:
        return "paramater is incorrect"

    try:
        f = open(file_path, 'w')
        f.write(time + "," + lux)
        return "succeeded to write"
    except Exception as e:
        print(e)
        return "failed to write"
    finally:
        f.close()
```

この部分は、ゲートウェイからのセンサデータをHTTPのPOSTメソッドで受け取り、CSVファイルに保存する処理である。

サーバが起動している状態で、以下のcURLコマンドを入力

```
curl -X POST -d "time=2019/02/24 22:47:25,&lux=700" localhost:17123/lux
```

すると、server.pyと同じディレクトリにsensor\_data.csvファイルが作成され(すでに存在する場合は更新)、時間と照度のデータが保存される。

上記のcURLコマンドの場合だと、以下のファイル内容になる。

sensor\_data.csv

```
2019/02/24 22:47:25,700 ,700
```

1列目が時刻,2列目が照度値である。

## 照度データ取得部分

```
@app.route('/lux', methods=['GET'])
def get_lux():
    try:
        f = open(file_path, 'r')
        for row in f:
            lux = row
        return lux
    except Exception as e:
        print(e)
        return e
    finally:
        f.close()
```

この部分では,localhost:17123/lux にHTTPのGETメソッドでリクエストを送ると,  
sensor\_data.csvに書き込まれている時間と照度データを返信するプログラムである.

例えば, 以下のcURLコマンドを入力すると

```
curl localhost:17123/lux
```

以下の内容が返信され,表示される.

```
2019/02/24 22:47:25,700
```

本手順書ではこれを利用して,ブラウザに照度データの表示を行う.

## index.html解説

index.htmlの解説を行う.

index.htmlの下記の部分の解説を行う.

```
<body onLoad="updateLux('/lux', 'data')">
  <div id = "data">test</div>
  <input type="button" value="更新" onclick="updateLux('/lux', 'data')">
</body>
```

onLoadはhtmlが読み込まれた時に実行される関数を登録し,  
onClickはボタンがクリックされた時に実行される関数を登録する.

つまり,このhtmlは読み込み時とボタンがクリックされた時にupdateLuxの関数を実行する.

次に,updateLuxの内容を記述している,index.htmlのJavaScriptの部分の解説を行う.

```
<script>
const updateLux = async(url, element) => {
  const sensorData = await fetch(url)
    .then(response => response.text())
  const target = document.getElementById(element)
  target.innerHTML = `<h1>${sensorData}</h1>`
}
</script>
```

この部分では,urlとelementを引数とし,urlからセンサデータを取得が,elementの内容を変更する関数である.

html内での使われかたで解説すると,  
urlは/luxなので,この関数は,localhost:17123/lux にGETメソッドでアクセスし, 時間と照度データの内容を取得し,sensorDataに代入する. その後,elementの内容,つまりdataエレメント(<div id = "data"></data>で囲まれた部分)の内容をsensorDataの内容に書き換えるものである.

## 最終結果

最終的に,この手順を終えると,以下のディレクトリ構成になっている.

```
server
├── sensor_data.csv
├── server.py
├── templates
│   └── index.html
```

## GitとGitHubへの登録

また,この作業内容をGitとGitHubに保存する. serverディレクトリで以下のコマンドを入力

```
git add .
git commit -m "サーバプログラム完成"
```

その後ブラウザで,GitHubにログイン,  
Start a project をクリック,  
以下の画像の内容を入力し,Create repositoryをクリック.

### Create a new repository

A repository contains all project files, including the revision history.

Owner

Repository name \*

 yamazki ▾

/ server ✓

Great repository names are short and memorable. Need inspiration? How about **congenial-bassoon**?

Description (optional)

☒  **Public**

Anyone can see this repository. You choose who can commit.

☐  **Private**

You choose who can see and commit to this repository.

☐ **Initialize this repository with a README**

This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore: **None** ▾

Add a license: **None** ▾



Create repository



クリック後,いくつかの作業内容が記述されているので, 其中的,  
...or push an existing repository from the command line 下に記されている作業内容のコマンドを実行.  
入力すると,GitHubのアカウントとパスワードの入力が求められるので入力.  
入力後,ページを更新すると,以下のような画面になっている.

[Manage topics](#)

1 commit

1 branch

0 releases

0 contributors

Branch: master

New pull request

Create new file

Upload files

Find file

Clone or download

Yamazaki and Yamazaki サーバプログラム完成

Latest commit f95cdf7 a minute ago

templates	サーバプログラム完成	a minute ago
.DS_Store	サーバプログラム完成	a minute ago
sensor_data.csv	サーバプログラム完成	a minute ago
server.py	サーバプログラム完成	a minute ago
server.pyc	サーバプログラム完成	a minute ago

Help people interested in this repository understand your project by adding a README.

Add a README

この手順を踏むことにより,GitHubへのプロジェクトの登録が完了する.

以後,このプロジェクトのコードを変更した場合には,

```
git add .  
git commit -m "変更内容"
```

とコミットした後に,

```
git push
```

とすれば,GitHubへのプロジェクトの変更内容を保存することができる.

## 2. センサデバイス作成

次に,ゲートウェイのプログラムを作成する.

```
void setup() {  
  // シリアルポートを開き, 通信速度を9600bpsに設定  
  Serial.begin(9600);  
}  
  
void loop() {  
  // 1000ms(1秒待ち)  
  delay(1000);  
}
```

```
// シリアル通信でtestを改行ありで出力(改行なしの場合はSerial.print)
Serial.println("test");
}
```

```
int val=0; //入力される値を格納する為の変数
void setup() {
  Serial.begin(9800); //モニターに出力するための設定
}
void loop() {
  //ANALOG INの0番ピンからデータを受け付ける
  val=analogRead(0);
  Serial.println(val/4); //入力された値をモニターに出力
  delay(100);
}
```

インストールするソフトウェアは,

# 付録

---

## node.js を利用したサンプルプログラム

本章では,[node.js](#)というサーバーサイドJavaScriptを利用したプログラムのサンプルを記載する.  
node.jsの使い方

1. 公式サイトでnode.jsをインストールする
2. ターミナルを起動,mkdirなどでプロジェクトディレクトリの作成
3. プロジェクトのルートディレクトリに移動
4. npm init というコマンドを入力し,プロジェクトの初期化を行う

上記の手順を行うことにより,node.jsアプリケーションを開発する準備が完了する.

また,手順書のサーバプログラムserver.pyをnode.jsのプログラムに置き換えたserver.jsを以下に記す.

また,このプログラムはexpressというフレームワークを利用しているので,それをインストールするために以下のコマンドを入力する

```
npm install -S express
```

```
const express = require("express");
const app = express();
const bodyParser = require('body-parser');
app.use(bodyParser.urlencoded({ extended: false }));
app.use(bodyParser.json());
const {promisify} = require('util');
const fs = require('fs');
const port = 17123;

// htmlのreturn
app.get('/', (req, res) => {
  res.sendFile('./index.html');
});

app.route('/lux')

  // luxデータ取得
  .get(async (req, res) => {
    const lux = await promisify(fs.readFile)
      ( './sensor_data.csv', {encoding : 'utf8'})
      .then(fileData => fileData.split(",")[1])
      .catch(err => res.send(err));

    res.send(lux);
  })

  // luxデータ書き込み(上書き)
  .post(async (req, res) => {
    res.setHeader('Content-Type', 'text/plain');
    const id = req.body.id;
    const lux = req.body.lux;
    promisify(fs.writeFile)
      ( './sensor_data.csv', id + "," + lux, 'utf-8')
      .then(() => res.send("succeeded to write file"))
      .catch(err => {
        console.log(err);
        res.send("failed to write file");
      });
  });
});
```

なお、この付録に記したプログラムは以下のURLにも記載している。

<https://github.com/yamazki/IoT-App/tree/master/node>