SensorFacade Documentation

Release 1.0

SensorFacade

Jan 06, 2018

Contents

- Link to github: https://github.com/ThomasDevoogdt/SensorFacade.git
- Link to docs: http://sensorfacade.readthedocs.io/en/latest/?badge=master

This library provides a facade for sensor data. It provides a way to attach, initialize, update and read the data.

Contents:

CHAPTER 1

SensorFacade

- Link to github: https://github.com/ThomasDevoogdt/SensorFacade.git
- Link to docs: http://sensorfacade.readthedocs.io/en/latest/?badge=master

This library provides a facade for sensor data. It provides a way to attach, initialize, update and read the data. New sensors should inherit from the **Sensor** class. If the sensor needs periodic attention or the read out takes a long time, implement the *update* function. If the sensor is just a simple analog reader, implement the *getData* function. Each sensors is provided with a name so it can directly used for e.g. a MQTT topic. Sensors with multiple metrics should inherit from **SensorSet** with contains a set of sensors

1.1 TODO

- Port to arduino-IDE
- Add sensors

Note: I'm working on Ubuntu 16.04 with PlatformIO, version 3.4.1 and GCC version 5.4.0

1.2 Install

```
pio lib install https://github.com/ThomasDevoogdt/SensorFacade.git
```

1.3 platformio.ini

If you're trying to compile sensor_bmp, add all the necessary dependencies

```
[env:d1_mini]
platform = espressif8266
board = d1_mini
framework = arduino

lib_deps =
    Adafruit BMP085 Unified
    Adafruit Unified Sensor
    Wire
```

1.4 Usage

```
//
// Created by thomas on 25/11/17.
//

#include <SensorFacade.h>
#include "CostumSensorHolder.h"
#include "CostumSensor.h"

SensorFacade sensorFacade = SensorFacade();

void setup() {
    Serial.begin(115200);
    Serial.println("Starting Up");

    auto *costumSensorHolder = new CostumSensorHolder();
    costumSensorHolder->registerCostumSensor1(new Sensor("sensor_holder_1"));
    costumSensorHolder->registerCostumSensor2(new Sensor("sensor_holder_2"));
    costumSensorHolder->registerCostumSensor3(new Sensor("sensor_holder_3"));
    sensorFacade.addSensorSet(costumSensorHolder);

    sensorFacade.addSensor(new CostumSensor("sensor"));

    sensorFacade.setSensorItr([](String name, Data data) {
        //data.value - data.time
        Serial.print("Sensor: " + name);
        Serial.println("Data: " + String(data.value));
    });

    sensorFacade.begin();
}

void loop() {
    // update sensor values
    sensorFacade.update();

    // iterate over sensors
    sensorFacade.ItrSensor();

    delay(1000);
}
```

1.5 Add single sensor

```
//
// Created by thomas on 25/11/17.
//

#ifndef SENSORFACADE_COSTUMSENSOR_H
#define SENSORFACADE_COSTUMSENSOR_H

#include "Arduino.h"
#include "SensorFacade.h"

class CostumSensor : public Sensor {
private:
public:
    CostumSensor(String name) : Sensor(name) {
        //
    }

    // when sensor needs periodic attention, don't impliment getData
    void update() override {
        data.value = NAN;
        data.time = timeProvider->getTime();
    }

    // or

    // in this case don't impliment update()
    Data getData() override {
        data.value = NAN;
        data.time = timeProvider->getTime();
        return data;
    }
};

#endif //SENSORFACADE_COSTUMSENSOR_H
```

1.6 Add sensor group

```
//
// Created by thomas on 25/11/17.
//

#ifndef SENSORFACADE_COSTUMSENSORHOLDER_H
#define SENSORFACADE_COSTUMSENSORHOLDER_H

#include "Arduino.h"
#include "SensorFacade.h"

class CostumSensorHolder : public SensorSet {
private:
    // pointers for ease use
    Sensor *costumSensor1;
    Sensor *costumSensor2;
    Sensor *costumSensor3;
```

```
public:
    explicit CostumSensorHolder() {

    };

    void registerCostumSensor1(Sensor *costumSensor1) {
        this->costumSensor1 = costumSensor1; // direct pointer
        this->addSensor(costumSensor1); // register in sensorSet
    }

    void registerCostumSensor2(Sensor *costumSensor2) {
        this->costumSensor2 = costumSensor2; // direct pointer
        this->addSensor(costumSensor2); // register in sensorSet
    }

    void registerCostumSensor3(Sensor *costumSensor3) {
        this->costumSensor3 = costumSensor3; // direct pointer
        this->addSensor(costumSensor3); // register in sensorSet
    }

    void begin() {

    }

    void update() {
        if (costumSensor1 != nullptr) {
            setSensorData(costumSensor1, Data(
                    1,
                    timeProvider->getTime()));
        }

        if (costumSensor2 != nullptr) {
            setSensorData(costumSensor2, Data(
                    2,
                    timeProvider->getTime()));
        }

        if (costumSensor3 != nullptr) {
            setSensorData(costumSensor3, Data(
                    3,
                    timeProvider->getTime()));
        }
    }
};

#endif //SENSORFACADE_COSTUMSENSORHOLDER_H
```

CHAPTER 2

Class Documentation

2.1 SensorFacade

class SensorFacade

The sensor facade holds a set of sensors and/or sensorSets.

Add all the sensors you'll keep track on and call the *begin()* function. Thereafter call periodically the *update()* function. As last provide a callback and call *ItrSensor()* to read them out.

Inherits from *ISensor*

Public Functions

SensorFacade()

creates a sensor facade instance

SensorFacade(*ITimeProvider* **timeProvider*)

creates a sensor facade instance with a time provider

Parameters

- `timeProvider`:

SensorFacade(void (**sensorItr*)) String, *Data*

creates a sensor facade instance with a iteration callback

Parameters

- `sensorItr`:

SensorFacade(*ITimeProvider* **timeProvider*, void (**sensorItr*)) String, *Data*

creates a sensor facade with a time provider and a iteration callback

Parameters

- `timeProvider`:
- `sensorItr`:

void **addSensor**(*Sensor* **sensor*)
add a sensor

Parameters

- `sensor`:

void **addSensorSet**(*SensorSet* **sensorSet*)
add a sensor set

Parameters

- `sensorSet`:

Sensor ***getSensor**(int *index*)
get a sensor by index

Return sensor

Parameters

- `index`:

void **setSensorItr**(void (**sensorItr*)) String, *Data*
set a callback for the sensor iteration

Parameters

- `sensorItr`:

void **ItrSensor**()
iterate over all the sensors using the predefined callback

int **size**()
get number of sensors

Return

void **update**()
call update function of all sensors

void **begin**()
call begin function of all sensors

2.2 SensorSet

class SensorSet

A sensor set helps you to add sensor metrics to one container.

e.g. a sensor that can reads the temperature as well the pressure.

Inherits from *ISensor*

Public Functions

SensorSet()

creates a sensor set

SensorSet(*ITimeProvider* *timeProvider)

creates a sensor set with a time provider

Parameters

- `timeProvider`:

void **begin**()

call begin function of all sensors

void **update**()

call update function of all sensors

SensorLinkedList<*Sensor* *> **getSensors**()

Return all the sensors of the container

2.3 Sensor

class Sensor

Each metric of each sensor should have an sensor implementation.

Implement the the update method or override the getData method.

Inherits from *ISensor*

Public Functions

Sensor(String *name*, *ITimeProvider* *timeProvider)

Parameters

- `name`: of the sensor instance
- `timeProvider`:

Sensor(String *name*)

Parameters

- `name`: of the sensor instance

Data **getData**()

Return last captured data

String **getName**()

Return name of the sensor instance - used for e.g. MQTT topic

2.4 ISensor

class ISensor

an abstract sensor class

Subclassed by *Sensor*, *SensorFacade*, *SensorSet*

Public Functions

virtual void **update**()

use for sensors that needs periodic attention

virtual void **begin**()

use for sensors that needs initialization

2.5 ITimeProvider

class ITimeProvider

an abstract time provider class

Public Functions

virtual unsigned long **getTime**()

Return abstract version returns always NAN

2.6 Data

struct Data

Public Functions

Data()

default struct constructor

Data(float *value*, unsigned long *time*)

Parameters

- `value`:
- `time`:

Public Members

float **value** = NAN
: data enum value

unsigned long **time** = NAN
: data enum time

This documentation was built using ArduinoDocs.

Index