Beehive Project

Generated by Doxygen 1.8.13

Contents

Chapter 1

Main Page

Real-Time Beehive Monitoring

We want to help beekeepers to keep an eye on their bees. By giving them insight on critical measurements we can monitor different events such as too low temperature, too high moisture, sufficient weight to harvest honey, as well as giving an estimate of how many bees went out of the beehive harvesting.

Monitoring your beehive and get real-time measurements on the website

This project is done in partnership with the Glasgow University Beekeeping Society

Table of contents

Setup

Instructions for installation can be found on the installation wiki.

General info

This projects architecture diagram:

Dependency graph for the C++ code:

Technologies

Project is created with:

- Raspberry pi 3.0
- Sensors
  - ADC101C021 ADC
  - HIH6131 Temperatur & Humidity x2
  - MPL115A2 Barometer
  - P82B96 I2C Shifter
- Node Js/express Js server
- MySQl database.

Authors

- **Remy Chatel, 2411062** - `Github`
- **Maysara Alhindi, 2417665** - `Github`
- **Trine Ødegård Olsen, 2420036** - `Github`

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 4

File Index

4.1 File List

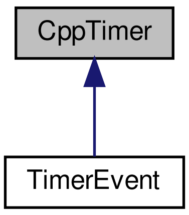
Here is a list of all documented files with brief descriptions:

Chapter 5

Class Documentation

5.1 CppTimer Class Reference

Inheritance diagram for CppTimer:



Public Member Functions

- void **start** (long nanosecs)
- virtual void **timerEvent** ()=0

Static Private Member Functions

- static void **handler** (int sig, siginfo_t ∗si, void ∗uc)

Private Attributes

- timer_t **timerid**
- struct sigevent **sev**
- struct sigaction **sa**
- struct itimerspec **its**

The documentation for this class was generated from the following file:

- CppTimer.h

5.2 Fan Class Reference

A class to control Fan.

```
#include <Fan.hpp>
```

Collaboration diagram for Fan:

Fan
saved_Fan_pointer

Public Member Functions

- Fan (int gpio=18, int interruptPin=22)

  A constructor.
- void setPwm (int pwm_value)

  Set the PWM value to the fan.
- void start ()

  Start the Fan.
- void stop ()

  Stop the Fan.
- ∼Fan ()

  A destructor.

Static Public Member Functions

- static void onInterrupt ()

  Static function triggers when the pin toggles.

Private Member Functions

- void setFanPointer ()

  Set the static Fan pointer saved_Fan_pointer.

Private Attributes

- int gpio

  A private variable.
- int interruptPin

  A private variable.

Static Private Attributes

- static Fan ∗ saved_Fan_pointer = NULL

  A private static variable.

5.2.1 Detailed Description

A class to control Fan.

Set different values for the Fan. Using a trigger pin connected to the humidity sensor inside the beehive, if the pin toggles (have threshold inside the sensor) it will trigger the onInterrupt() in this class and start or stop the Fan depending on if the pin goes to high or low.

5.2.2 Constructor & Destructor Documentation

5.2.2.1 Fan()

```
Fan::Fan (
            int gpio = 18,
            int interruptPin = 22 )
```

A constructor.

Set the private variables, default values is implemented

Parameters

gpio	an integer giving the gpio pin the fan is connected.
interruptPin	an integer giving the gpio pin the alarmpin is connected.

5.2.2.2 ∼Fan()

```
Fan::∼Fan ( )
```

A destructor.

Stops the Fan

5.2.3 Member Function Documentation

5.2.3.1 onInterrupt()

```
void Fan::onInterrupt ( )  [static]
```

Static function triggers when the pin toggles.

Static to be called in real-time If the pin goes from high to low the stop() function is called If the pin goes from low to high the start() function is called

See also

start() and stop()

5.2.3.2 setFanPointer()

```
void Fan::setFanPointer ( )  [private]
```

Set the static Fan pointer saved_Fan_pointer.

Called by the constructor. Saves the objects in the corresponding private variables.

See also

Fan(int gpio = 18, int interruptPin=22), saved_Fan_pointer()

5.2.3.3 setPwm()

```
void Fan::setPwm (
             int pwm_value )
```

Set the PWM value to the fan.

The PWM range is 0-1023.

5.2.3.4 start()

```
void Fan::start ( )
```

Start the Fan.

Calls the setPWM(1023) to maximise the Fan

See also

setPwm(int pwm_value)

5.2.3.5 stop()

```
void Fan::stop ( )
```

Stop the Fan.

Calls the setPWM(0) to stop the Fan

See also

setPwm(int pwm_value)

5.2.4 Member Data Documentation

5.2.4.1 gpio

```
int Fan::gpio  [private]
```

A private variable.

Integer to hold the gpio pin the Fan is connected.

5.2.4.2 interruptPin

```
int Fan::interruptPin  [private]
```

A private variable.

Integer to hold the gpio pin the alarmpin form humidity sensor is connected.

5.2.4.3 saved_Fan_pointer

```
Fan * Fan::saved_Fan_pointer = NULL  [static], [private]
```

A private static variable.

Assuming we only have one fan to trigger. Need a static pointer to the object so that the static interrupt function know which fan to set. If we have multiple fans on different pins, this has to be changed to a list/vector/pointer.

The documentation for this class was generated from the following files:

- Fan.hpp
- Fan.cpp

5.3 I2C Class Reference

A I2C class handling the I2C bus.

```
#include <I2C.hpp>
```

Public Member Functions

- I2C (char ∗portI2C, int slaveAddr)

  A constructor.
- void readI2C (int bytesToRead, unsigned char ∗global_buffer)

  Read data from the peripheral .
- int writeI2C (int length, unsigned char ∗buffer)
- int writeI2C (int length)

  Write data to the peripheral .

Private Attributes

- int file_i2c

  A private variable.
- unsigned char buffer [5] = {0}

  A private variable.

5.3.1 Detailed Description

A I2C class handling the I2C bus.

Handling I2C bus intended to allow multiple "slave" digital chips to communicate with the master i.e Raspberry PI. Every sensor connected to the bus has their own I2C-object.

5.3.2 Constructor & Destructor Documentation

5.3.2.1 I2C()

```
I2C::I2C (
            char ∗ portI2C,
            int slaveAddr )
```

A constructor.

Open the I2C bus and tell the kernel the I2C address of the slave.

Parameters

portI2C	a char pointer holding the which I2C bus on the Raspberry the peripheral is connected to.
slaveAddr	an integer holding the address to the peripheral.

5.3.3 Member Function Documentation

5.3.3.1 readI2C()

```
void I2C::readI2C (
            int bytesToRead,
            unsigned char * global_buffer )
```

Read data from the peripheral .

Parameters

bytesToRead	an integer giving numbers of bytes to read.
global_buffer	an unsigned char pointer where the data read is saved.

5.3.3.2 writeI2C() [1/2]

```
int I2C::writeI2C (
            int length,
            unsigned char * buffer )
```

Parameters

length	an integer giving numbers of bytes to write.
buffer	an unsigned char pointer pointing to the data to write.

5.3.3.3 writeI2C() [2/2]

```
int I2C::writeI2C (
            int length )
```

Write data to the peripheral .

Using the private buffer in the object to write default value

Parameters

length	an integer giving numbers of bytes to write.

5.3.4 Member Data Documentation

5.3.4.1 buffer

```
unsigned char I2C::buffer[5] = {0}  [private]
```

A private variable.

Used if you want to write to a peripheral, starting a conversation

5.3.4.2 file_i2c

```
int I2C::file_i2c  [private]
```

A private variable.

To communicate with the I2C bus

The documentation for this class was generated from the following files:

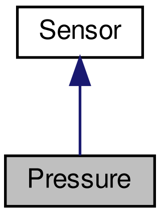
- I2C.hpp
- I2C.cpp

5.4 Pressure Class Reference

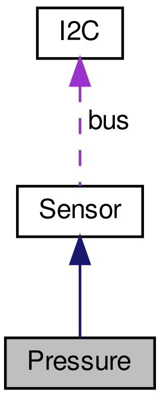
A class for the MPL115A2 Barometer sensor.

```
#include <Pressure.hpp>
```

Inheritance diagram for Pressure:



Collaboration diagram for Pressure:



Public Member Functions

- Pressure (char ∗portI2C, int addrI2C, unsigned char ∗global_buffer)

    A constructor.
- void readCoefficients (unsigned char ∗global_buffer)

    Call the I2C-object write and read function to get sensor coefficients.
- void readI2C (unsigned char ∗global_buffer)

    Call the I2C-object read function.
- int writeI2C ()

    Call the I2C-object write function.
- float getPressure ()

    Convert the humidity bits into kPa.
- float getTemp ()

    Convert the temperature bits into degrees.
- ∼Pressure ()

    A destructor.

Private Attributes

- float a0

    A private variable.
- float b1

    A private variable.
- float b2

    A private variable.
- float c12

    A private variable.
- int pressure

    A private variable.
- int temp

    A private variable.
- float pressureComp

    A private variable.

Additional Inherited Members

5.4.1 Detailed Description

A class for the MPL115A2 Barometer sensor.

Handling communication with the sensor, and converting the sensor measurements to human readable values

5.4.2 Constructor & Destructor Documentation

5.4.2.1 Pressure()

```
Pressure::Pressure (
              char * portI2C,
              int addrI2C,
              unsigned char * global_buffer )
```

A constructor.

Call the Sensor constructor and readCoefficients(unsigned char∗ global_buffer)

Parameters

portI2C	a char pointer holding the which I2C bus on the Raspberry the peripheral is connected to.
addrI2C	an integer holding the address to the peripheral.

See also

Sensor(int bytesToRead, int bytesToWrite, char∗ portI2C, int addrI2C)

5.4.2.2 ∼Pressure()

```
Pressure::∼Pressure ( )
```

A destructor.

Empty, can be used if more functions are integrated

5.4.3 Member Function Documentation

5.4.3.1 getPressure()

```
float Pressure::getPressure ( )
```

Convert the humidity bits into kPa.

Returns

The humidity in kPa.

5.4.3.2 getTemp()

```
float Pressure::getTemp ( )
```

Convert the temperature bits into degrees.

Returns

The temperature in degrees.

5.4.3.3 readCoefficients()

```
void Pressure::readCoefficients (
            unsigned char * global_buffer )
```

Call the I2C-object write and read function to get sensor coefficients.

Write (uint8_t)(0x04) to the sensor, followed by a read to get the coefficients we need to decrypt the values from the pressure sensor. Saves the coefficients in the corresponding private variables.

Parameters

global_buffer	an unsigned char pointer where the data read is saved.

5.4.3.4 readI2C()

```
void Pressure::readI2C (
            unsigned char * global_buffer )  [virtual]
```

Call the I2C-object read function.

Get the data read form the sensor and split the data into pressure, temperature and calculate the pressure component bits and saves them to the corresponding private variable.

Parameters

global_buffer	an unsigned char pointer where the data read is saved.

Implements Sensor.

5.4.4 Member Data Documentation

5.4.4.1 a0

```
float Pressure::a0  [private]
```

A private variable.

variable to hold the floating point coefficient a0 needed to convert the read sensor bit to a human readable value

5.4.4.2 b1

```
float Pressure::b1  [private]
```

A private variable.

variable to hold the floating point coefficient b1 needed to convert the read sensor bit to a human readable value

5.4.4.3 b2

```
float Pressure::b2  [private]
```

A private variable.

variable to hold the floating point coefficient b2 needed to convert the read sensor bit to a human readable value

5.4.4.4 c12

```
float Pressure::c12  [private]
```

A private variable.

variable to hold the floating point coefficient c2 needed to convert the read sensor bit to a human readable value

5.4.4.5 pressure

```
int Pressure::pressure  [private]
```

A private variable.

Holding the newest pressure-bit reading from the sensor

5.4.4.6 pressureComp

```
float Pressure::pressureComp  [private]
```

A private variable.

Intermediate variable needed to convert the sensor data to a human readable value. Calculated by combining all the private variables

5.4.4.7 temp

```
int Pressure::temp  [private]
```

A private variable.

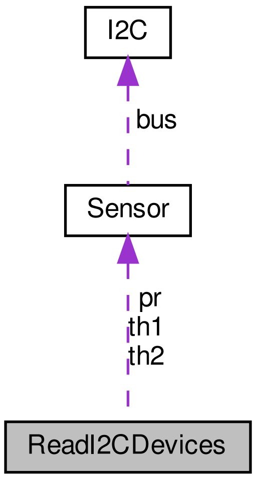
Holding the newest temperature-bit reading from the sensor

The documentation for this class was generated from the following files:

- Pressure.hpp
- Pressure.cpp

5.5 ReadI2CDevices Class Reference

Collaboration diagram for ReadI2CDevices:



Public Member Functions

- ReadI2CDevices ()

    A constructor.
- void writeAll ()

    Write to all the peripherals.
- std::string readAll ()

    Read data from all the peripherals.
- ∼ReadI2CDevices ()

    A destructor.

Private Member Functions

- void createSensorObjects ()

    Create all the sensor objects we want to communicate with.

Private Attributes

- unsigned char global_buffer [20] ={0}

    A private variable.
- Sensor ∗ th1

    A private variable.
- Sensor ∗ th2

    A private variable.
- Sensor ∗ pr

    A private variable.

5.5.1 Constructor & Destructor Documentation

5.5.1.1 ReadI2CDevices()

```
ReadI2CDevices::ReadI2CDevices ( )
```

A constructor.

Call the function createSensorObjects();

See also

createSensorObjects()

5.5.1.2 ∼ReadI2CDevices()

```
ReadI2CDevices::∼ReadI2CDevices ( )
```

A destructor.

Deallocates memory from heap

5.5.2 Member Function Documentation

5.5.2.1 createSensorObjects()

```
void ReadI2CDevices::createSensorObjects ( )  [private]
```

Create all the sensor objects we want to communicate with.

Called by the constructor. Saves the objects in the corresponding private variables.

See also

ReadI2CDevices();

5.5.2.2 readAll()

```
std::string ReadI2CDevices::readAll ( )
```

Read data from all the peripherals.

Call the pure virtual function readI2C(unsigned char∗ global_buffer) in Sensor.hpp for all the sensors. Sends the private global_buffer to save the readings.

Returns

A string of all the human readable data read form the sensors

See also

readI2C(unsigned char∗ global_buffer) = 0;

5.5.2.3 writeAll()

```
void ReadI2CDevices::writeAll ( )
```

Write to all the peripherals.

Call the virtual function writeI2C() in Sensor.hpp for all the sensors

See also

writeI2C()

5.5.3 Member Data Documentation

5.5.3.1 global_buffer

```
unsigned char ReadI2CDevices::global_buffer[20] ={0}  [private]
```

A private variable.

Global buffer where all the sensors saves the measurements.

5.5.3.2 pr

```
Sensor* ReadI2CDevices::pr  [private]
```

A private variable.

Sensor object, pointing to the pressure sensor on I2C bus 1.

5.5.3.3 th1

```
Sensor* ReadI2CDevices::th1  [private]
```

A private variable.

Sensor object, pointing to the temperature and humidity sensor on I2C bus 1.

5.5.3.4 th2

```
Sensor* ReadI2CDevices::th2  [private]
```

A private variable.

Sensor object, pointing to the temperature and humidity sensor on I2C bus 2.

The documentation for this class was generated from the following files:

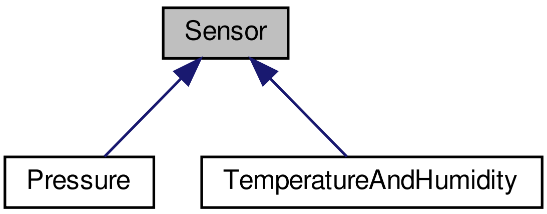
- ReadI2CDevices.hpp
- ReadI2CDevices.cpp

5.6 Sensor Class Reference

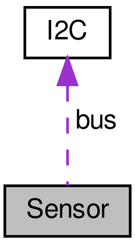
An abstract class.

```
#include <Sensor.hpp>
```

Inheritance diagram for Sensor:



Collaboration diagram for Sensor:



Public Member Functions

- Sensor (int bytesToRead, int bytesToWrite, char ∗portI2C, int addrI2C)

  A constructor.
- virtual void readI2C (unsigned char ∗global_buffer)=0

  A pure virtual member.
- virtual int writeI2C ()

  A virtual member.
- ∼Sensor ()

  A destructor.

Protected Attributes

- int bytesToRead

  A private variable.
- int bytesToWrite

  A private variable.
- I2C ∗ bus

  A private variable.

5.6.1 Detailed Description

An abstract class.

Used as a base class for sensors connected to the I2C bus of the RaspberryPi

5.6.2 Constructor & Destructor Documentation

5.6.2.1 Sensor()

```
Sensor::Sensor (
              int bytesToRead,
              int bytesToWrite,
              char * portI2C,
              int addrI2C )  [inline]
```

A constructor.

Create a new I2C object used to communicate with the sensor.

Parameters

bytesToRead	default bytes to read form the I2C bus
bytesToWrite	default bytes to write form the I2C bus
portI2C	a char pointer holding which I2C bus on the Raspberry the peripheral is connected to.
addrI2C	an integer holding the address to the peripheral.

See also

I2C(char∗ portI2C, int slaveAddr)

5.6.2.2 ∼Sensor()

```
Sensor::∼Sensor ( )  [inline]
```

A destructor.

Deallocates memory from heap

5.6.3 Member Function Documentation

5.6.3.1 readI2C()

```
virtual void Sensor::readI2C (
             unsigned char * global_buffer )  [pure virtual]
```

A pure virtual member.

Read data from the peripheral .

Parameters

global_buffer	an unsigned char pointer where the data read is saved.

Implemented in Pressure, and TemperatureAndHumidity.

5.6.3.2 writeI2C()

```
virtual int Sensor::writeI2C ( )  [inline], [virtual]
```

A virtual member.

Write data to the peripheral .

Reimplemented in Pressure, and TemperatureAndHumidity.

5.6.4 Member Data Documentation

5.6.4.1 bus

```
I2C* Sensor::bus  [protected]
```

A private variable.

Holding the sensor I2C bus object created in the constructor

5.6.4.2 bytesToRead

```
int Sensor::bytesToRead  [protected]
```

A private variable.

Holding the sensor default bytes to read form the I2C bus

5.6.4.3 bytesToWrite

`int Sensor::bytesToWrite [protected]`

A private variable.

Holding the sensor default bytes to write form the I2C bus

The documentation for this class was generated from the following file:

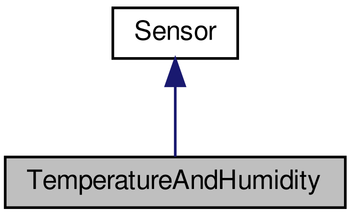
- Sensor.hpp

5.7 TemperatureAndHumidity Class Reference

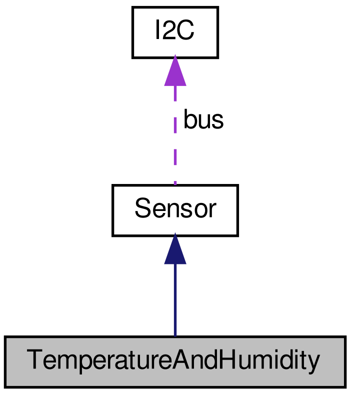
A class for the HIH6131 Temperatur & Humidity sensor.

`#include <TemperatureAndHumidity.hpp>`

Inheritance diagram for TemperatureAndHumidity:



Collaboration diagram for TemperatureAndHumidity:



Public Member Functions

- TemperatureAndHumidity (char ∗portI2C, int addrI2C)

  A constructor.
- void readI2C (unsigned char ∗global_buffer)

  Call the I2C-object read function.
- int writeI2C ()

  Call the I2C-object write function.
- unsigned int getStatus ()

  Convert the status bits into a integer from 0-3.
- double getTemp ()

  Convert the temperature bits into degrees.
- double getHum ()

  Convert the humidity bits into RH.
- ∼TemperatureAndHumidity ()

  A destructor.

Private Attributes

- uint8_t status

  A private variable.
- uint16_t humidity

  A private variable.
- uint16_t temperature

  A private variable.

Additional Inherited Members

5.7.1 Detailed Description

A class for the HIH6131 Temperatur & Humidity sensor.

Handling how to communicate with the sensor, and converting the sensor measurements to human readable values

5.7.2 Constructor & Destructor Documentation

5.7.2.1 TemperatureAndHumidity()

```
TemperatureAndHumidity::TemperatureAndHumidity (
              char * portI2C,
              int addrI2C )
```

A constructor.

Call the Sensor constructor

Parameters

portI2C	a char pointer holding the which I2C bus on the Raspberry the peripheral is connected to.
addrI2C	an integer holding the address to the peripheral.

See also

Sensor(int bytesToRead, int bytesToWrite, char∗ portI2C, int addrI2C)

5.7.2.2 ∼TemperatureAndHumidity()

```
TemperatureAndHumidity::∼TemperatureAndHumidity ( )
```

A destructor.

Empty, can be used if more functions are integrated

5.7.3 Member Function Documentation

5.7.3.1 getHum()

```
double TemperatureAndHumidity::getHum ( )
```

Convert the humidity bits into RH.

Returns

The humidity in RH.

5.7.3.2 getStatus()

```
unsigned int TemperatureAndHumidity::getStatus ( )
```

Convert the status bits into a integer from 0-3.

Returns

The status of the sensor.

5.7.3.3 getTemp()

```
double TemperatureAndHumidity::getTemp ( )
```

Convert the temperature bits into degrees.

Returns

The temperature in degrees.

5.7.3.4 readI2C()

```
void TemperatureAndHumidity::readI2C (
             unsigned char * global_buffer )  [virtual]
```

Call the I2C-object read function.

Get the data read form the sensor and split the data into status, humidity and temperatur bits and saves them to the corresponding private variable.

Parameters

global_buffer	an unsigned char pointer where the data read is saved.

Implements Sensor.

5.7.4 Member Data Documentation

5.7.4.1 humidity

```
uint16_t TemperatureAndHumidity::humidity  [private]
```

A private variable.

Holding the newest humidity-bit reading from the sensor

5.7.4.2 status

```
uint8_t TemperatureAndHumidity::status  [private]
```

A private variable.

Holding the newest status-bit reading from the sensor

5.7.4.3 temperature

```
uint16_t TemperatureAndHumidity::temperature  [private]
```

A private variable.

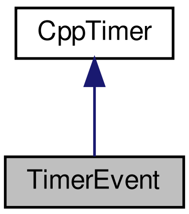
Holding the newest temperature-bit reading from the sensor

The documentation for this class was generated from the following files:

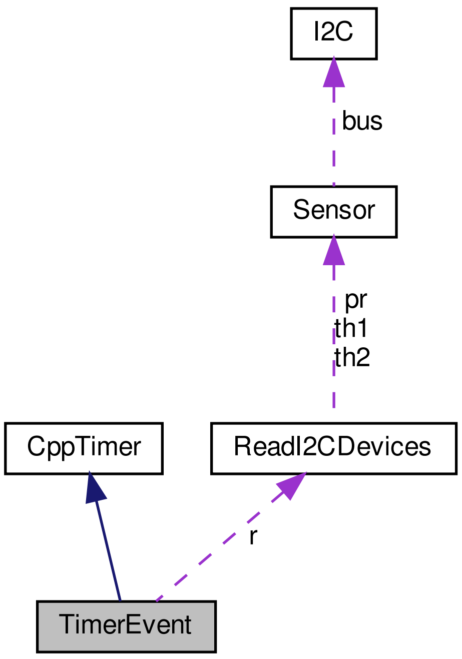
- TemperatureAndHumidity.hpp
- TemperatureAndHumidity.cpp

5.8 TimerEvent Class Reference

Inheritance diagram for TimerEvent:



Collaboration diagram for TimerEvent:



Public Member Functions

- TimerEvent ()

  A constructor.
- ∼TimerEvent ()

  A destructor.

Private Member Functions

- void timerEvent ()

  A private member.

Private Attributes

- ReadI2CDevices ∗ r

  A private variable.
- int sockfd

  A private variable.
- struct sockaddr_in servaddr

  A private variable.

5.8.1 Constructor & Destructor Documentation

5.8.1.1 TimerEvent()

```
TimerEvent::TimerEvent ( )
```

A constructor.

Create a new ReadI2CDevices-object and saves it to the corresponding private variable. Setup and configure the socket

5.8.1.2 ∼TimerEvent()

```
TimerEvent::∼TimerEvent ( )
```

A destructor.

Deallocates memory from heap

5.8.2 Member Function Documentation

5.8.2.1 timerEvent()

```
void TimerEvent::timerEvent ( ) [private], [virtual]
```

A private member.

Called by the CppTimer.h Get a string of all the sensor data and send it to the server

Implements CppTimer.

5.8.3 Member Data Documentation

5.8.3.1 r

```
ReadI2CDevices* TimerEvent::r [private]
```

A private variable.

Object to get all the gathered sensor data.

5.8.3.2 servaddr

```
struct sockaddr_in TimerEvent::servaddr  [private]
```

A private variable.

Struct containing the server address, port number and host number

5.8.3.3 sockfd

```
int TimerEvent::sockfd  [private]
```

A private variable.

The socket created in the constructor

The documentation for this class was generated from the following files:

- TimerEvent.hpp
- TimerEvent.cpp

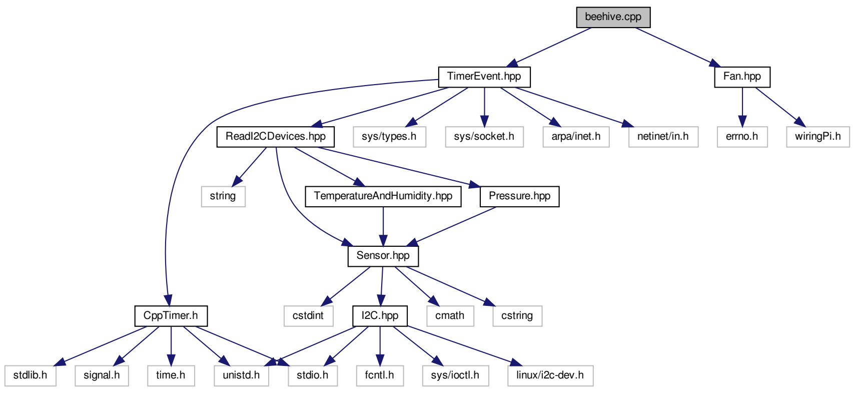
Chapter 6

File Documentation

6.1 beehive.cpp File Reference

```
#include "TimerEvent.hpp"
#include "Fan.hpp"
```

Include dependency graph for beehive.cpp:



Functions

- int main (int argc, const char ∗argv[ ])

6.1.1 Detailed Description

Project start

6.1.2 Function Documentation

6.1.2.1 main()

```
int main (
            int argc,
            const char * argv[] )
```

Create a Fan object so the interrupt bin is set to the default value (using the default gpio pin in the Fan constructor) Create a TimerEvent object to start the timer which trigger the sensor-readings.

Index