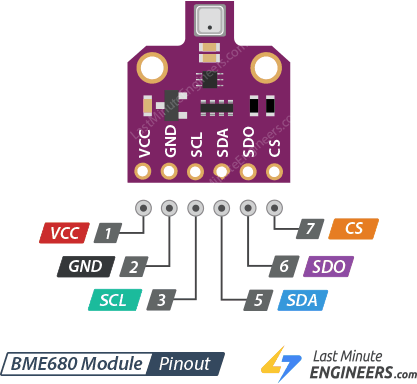
**BME680 Environmental Sensor with ESP32**

* **Overview of Project:**
* In this project, we will monitor the BME680 parameters () with the ESP32.
* We will use the advanced BSEC library for BME680 and monitor its parameters including IAQ on a web server. So, that we can monitor the sensor data.
* We will use Network Time Protocol (NTP) for getting the date and time and data sent on the server at a particular time and time set from the serial monitor of Arduino ide.
* **Overview of BME680 Sensor:**
* This sensor has sensing capabilities for temperature, humidity, barometric pressure, and VOC gas (volatile organic compounds).
* It contains a small MOx (metal-oxide) sensor. The heated metal oxide changes resistance based on VOCs in the air, so it can be used to detect gases & alcohols such as Ethanol, Alcohol, and Carbon Monoxide and perform air quality measurements.
* The heated metal oxide changes resistance based on VOCs in the air, so it can be used to detect gases & alcohols such as Ethanol, Alcohol, and Carbon Monoxide and perform air quality.
* The sensor operates from 1.7V to 3.6V. The standby power consumption of this module is 0.29 to 0.8 uA, while in sleep mode the power consumption is between 0.15 to 1 uA.



* BME680 can measure the Air quality index (IAQ) from 0 to 500 PPM.



* **Sensor’s working:**
* BME680 gas sensor is fabricated with micro-electro-mechanical system (MEMS) technology.
* MEMS-based gas detection provides dramatic size reduction, a reduction in power consumption, and the ability to increase functionality and selectivity.
* A typical MEMS gas sensor consists of a metal oxide semiconductor layer (sensing material) that reacts with the target gas, at least two electrodes to monitor the electrical resistance/current variations, and a micro-hotplate to increase the operating temperature.
* When the metal oxide semiconductor layer is heated at a high temperature, oxygen is absorbed on the surface.
* In clean air, electrons from the conduction band in metal oxide are attracted to oxygen molecules.
* This forms an electron depletion layer just below the surface of metal-oxide particles and forms a potential barrier.
* As a result, the metal-oxide layer becomes highly resistive and prevents electric current flow.
* In the presence of gases, however, the surface density of adsorbed oxygen decreases as it reacts with the gases; which lowers the potential barrier.
* Electrons are then released into the metal-oxide layer, allowing current to flow freely through the sensor.
* **Network Time Protocol (NTP):**
* Network Time Protocol (NTP) is a standard Internet Protocol (IP) for synchronizing the clocks of computers on a network over TCP/IP-based networks.
* NTP coordinates the real-time clocks of computers on a network Developed at the University of Delaware in 1985.
* It is one of the oldest internet protocols still in use and the latest release is version 4.2.
* Most operating systems, including [Linux](https://timetoolsltd.com/ntp/how-to-install-and-configure-ntp-on-linux/), macOS, and Windows have NTP implementations.
* NTP communicates using the User Datagram Protocol (UDP), port 123.
* Computer clocks can be coordinated within a few milliseconds of UTC.

## **How NTP Works:**

## NTP can operate in unicast mode, or client-server.

## This is where a client transmits a request packet to a server, which response with a time-stamped packet.

## Each packet has originated, receive and transmit timestamps so that network propagation delays can be calculated.

## This allows clients to synchronize as tightly as possible to the server’s clock.

## All time-stamp packets generated by NTP utilize UTC time and Coordinated Universal Time (UTC) is a worldwide time standard and closely related to GMT (Greenwich mean time).

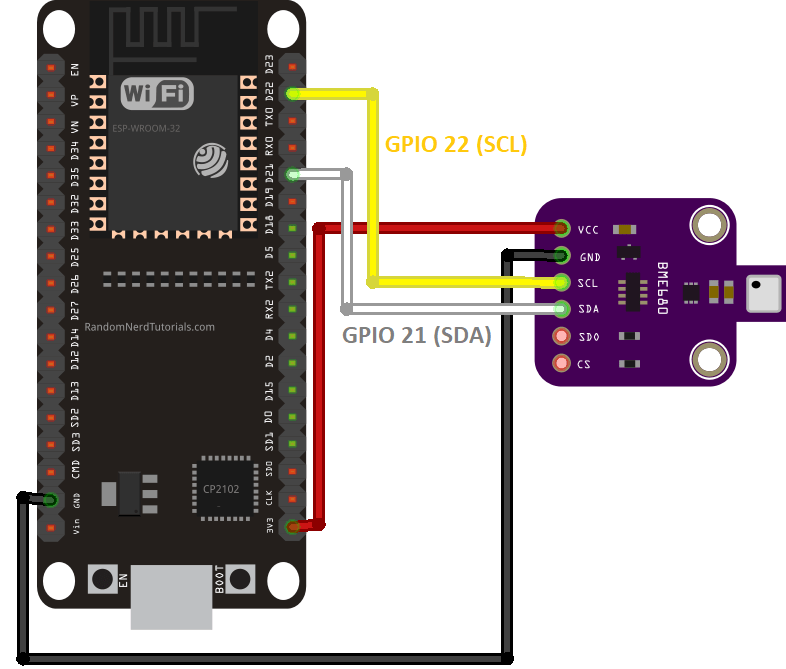
## UTC does not vary, it is the same worldwide. NTP sets the clocks of computers to UTC, any local time zone offset or daylight-saving time offset is applied by the operating system.

* **Digital interfaces:** I2C or SPI protocols can be interfaced, but I have used I2C protocols.

**Pins:** SCL- It is the SPI Clock pin, it’s an input to the chip.

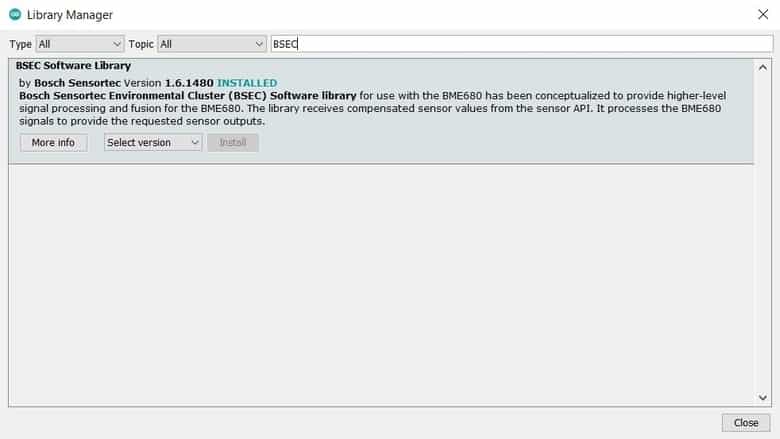
SDA- It is also the I2C data pin, connect to your controller I2C data line.

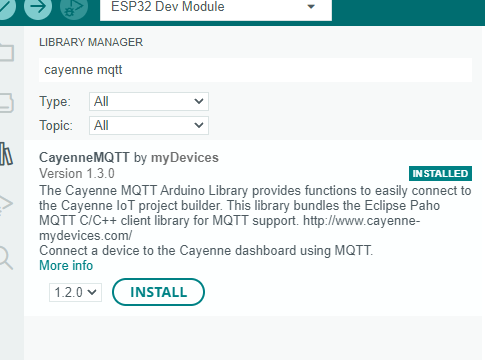
* **Connection:**



## **Library Installation:**

## To begin reading sensor data, we will need to install the **BSEC Library** for BME680. It is available from the Arduino library manager.



* To access the example sketches, navigate to the **File > Examples > BSEC BME680 Library** You will see a selection of example sketches.
* To data send on the cayenne server, we will need to install the **cayenne MQTT library**.  It is available from the Arduino library manager
* **Project link:** [BME680\_Project.zip](https://stemrobotechnologiespl-my.sharepoint.com/:u:/g/personal/ankita_agrhari_tevatrontech_in/EXPZYyJFEFRFgQewyYqz-4QBUeM3UiWErFGLC4JbMZ5xsA?e=YucCf1)
* **Programming Code Explanation:**
* **For Cayenne server:**

**Include libraries:** CayenneMQTTESP32.h

We need to enter Wi-Fi connection details in place of SSID and PASSWORD.

Char SSID [] = "replace with your SSID"

Char wifiPassword [] = "replace with your password”

we need to provide Cayenne with authentication information. This should be obtained from the Cayenne Dashboard.

char username [] = "MQTT user name

char password [] = "MQTT password”;

char ClientID [] = "MQTT clint";

* **For the NTP server:**

**Include libraries: time.h**

1. **NTP Server and Time Settings:** We need to define the following variables to configure and get time from an NTP server:
2. **NTP Server:** We’ll request the time from ***pool.ntp.org***, which is a cluster of timeservers that anyone can use to request the time.

const char\* variable = "pool.ntp.org"

1. **GMT Offset**: The gmtOffset\_sec variable defines the offset in seconds between your time zone and GMT, we live in Portugal, so the time offset is 0 and change the time gmtOffset\_sec variable to match your time zone**.**

// set Indian time zone

const long variable = 3600\*5+1800

1. **Daylight Offset:** The daylightOffset\_sec variable defines the offset in seconds for daylight saving time and It is generally one hour, which corresponds to 3600 seconds.

const int variable = 3600

create a time structure (struct tm) called time info that contains all the details about the time (min, sec, hour, etc…).

struct tm time info;

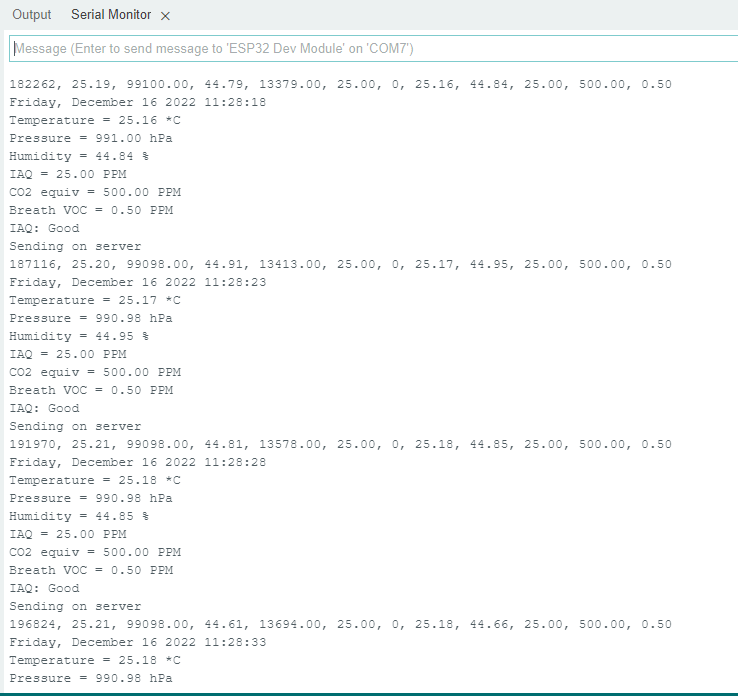
The tm structure contains a calendar date and time broken down into its components:

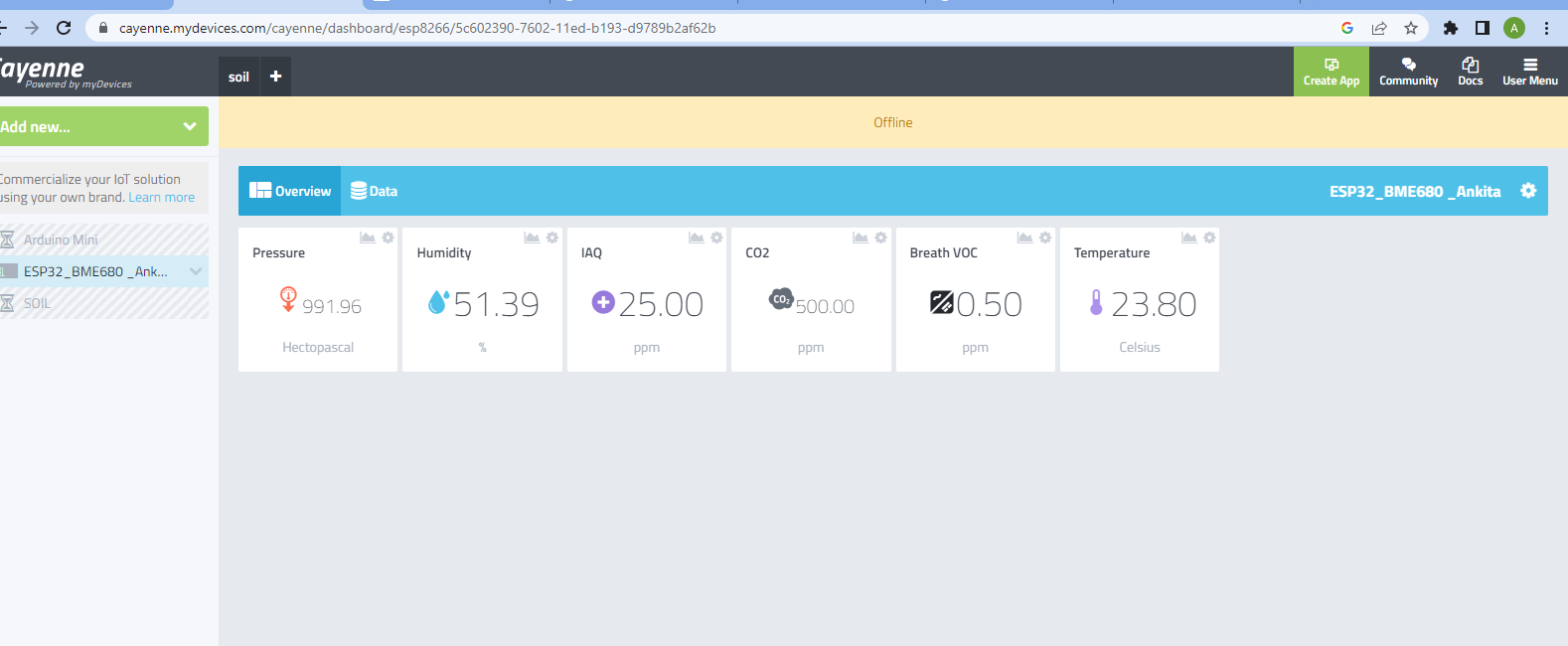
tm sec: seconds after the minute;

tm min: minutes after the hour;

tm hour: hours since midnight;

tm day: day of the month;

* **For Serial Monitor:** We will set a time from the serial monitor of Arduino ide in {hours: Minutes} formats.
* **Output:**
* **Serial Monitor:**
* **Cayenne Server Output**

1. **Dashboard output**
2. **Live data**