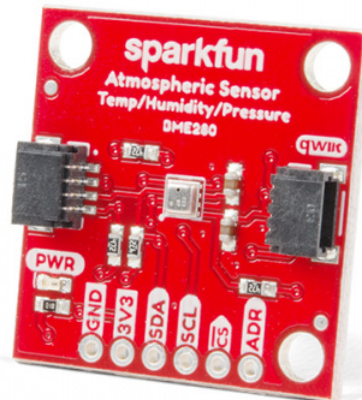


Qwiic Atmospheric Sensor (BME280) Hookup Guide

Introduction

The new Qwiic Atmospheric Sensor (BME280) is an updated board revision of our Atmospheric Sensor Breakout-BME280 to make it Qwiic compatible. The Qwiic connector system reduces the hassle of interfacing to the sensor via I²C, by utilizing polarized cables that are simple to use. The BME280 is great for measuring humidity, temperature, and barometric pressure.



SparkFun Atmospheric Sensor Breakout - BME280 (Qwiic)

● SEN-15440

In addition, we now provide a Python library for compatibility with single board computer (SBC) platforms like the Raspberry Pi boards. The Arduino library is shared from the preexisting hardware. The examples below, will demonstrate how to use the Qwiic Atmospheric Sensor with a RedBoard Qwiic; and a Raspberry Pi with the Qwiic pHAT (and additional accessories) utilizing the Qwiic connection system.

Don't forget to check out this great video of Rob playing his **sparxophone** thanks to the help of the BME280!

Product Showcase: SparkFun Atmospheric Sensor Breakout



Required Materials

The Qwiic Atmospheric Sensor does need a few additional items for you to get started. The RedBoard Qwiic is for the Arduino examples and the Qwiic pHAT is for the Raspberry Pi example (see note below). You may already have a few of these items, so feel free to modify your cart based on your needs. Additionally, there are also alternative parts options that are available as well (*click button below to toggle options*).



SparkFun RedBoard Qwiic

🛒 DEV-15123



SparkFun Qwiic Cable Kit

🛒 KIT-15081



SparkFun Qwiic pHAT for Raspberry Pi

🛒 DEV-15351

ALTERNATIVE PARTS (TOGGLE)

Raspberry Pi Example: If you don't already have them, you will need a Raspberry Pi and standard peripherals. An example setup is listed below. *(This sensor and the Python library have not been tested on the newly released Raspberry Pi 4 because we don't carry it in our catalog yet.)*



Raspberry Pi 3 B+
● DEV-14643



pi-topCEED (Green)
● KIT-14035



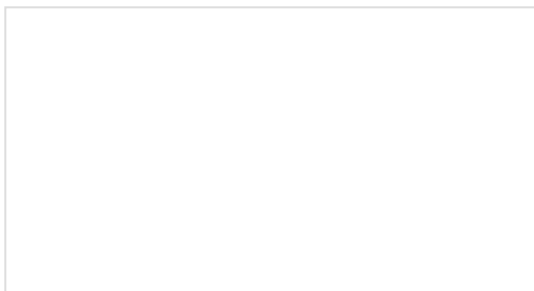
Multimedia Wireless Keyboard
● WIG-14271



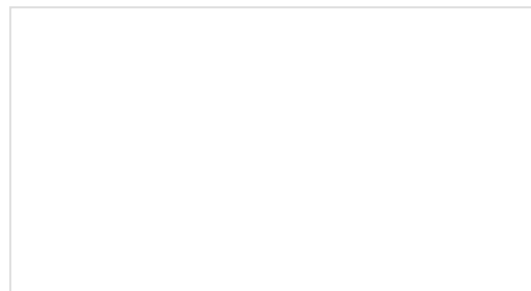
Raspberry Pi™ - 16GB MicroSD NOOBS Card
● COM-13945

Suggested Reading

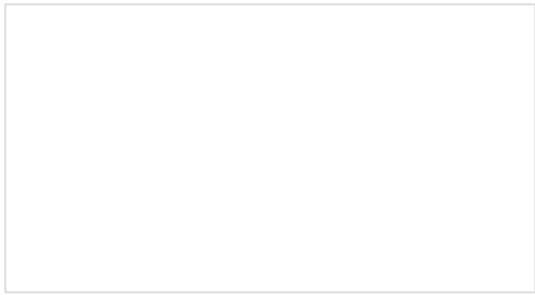
If you're unfamiliar with jumper pads, I²C, Qwiic, or Python be sure to checkout some of these foundational tutorials. Also included, in this list, are past tutorials involving the BME280 sensor.



Serial Peripheral Interface (SPI)
SPI is commonly used to connect microcontrollers to peripherals such as sensors, shift registers, and SD cards.

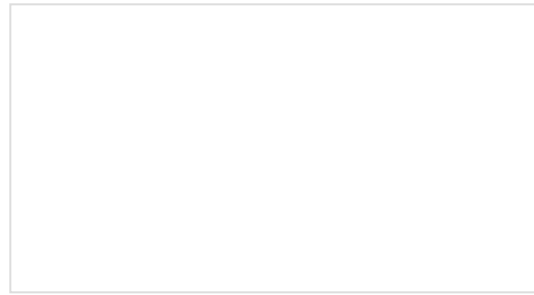


Logic Levels
Learn the difference between 3.3V and 5V devices and logic levels.



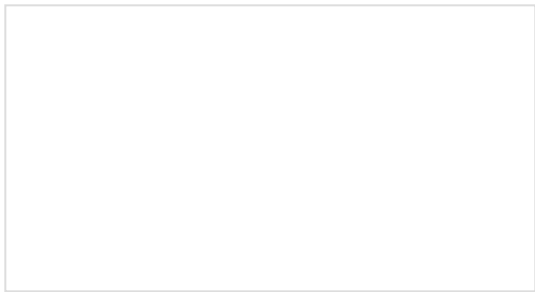
I2C

An introduction to I2C, one of the main embedded communications protocols in use today.



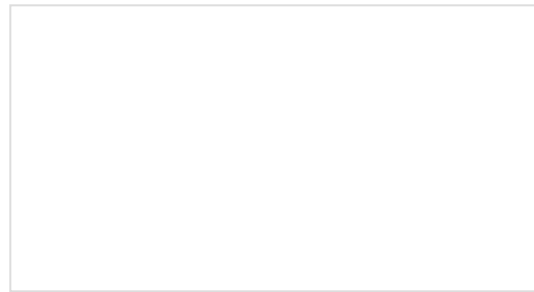
How to Work with Jumper Pads and PCB Traces

Handling PCB jumper pads and traces is an essential skill. Learn how to cut a PCB trace, add a solder jumper between pads to reroute connections, and repair a trace with the green wire method if a trace is damaged.



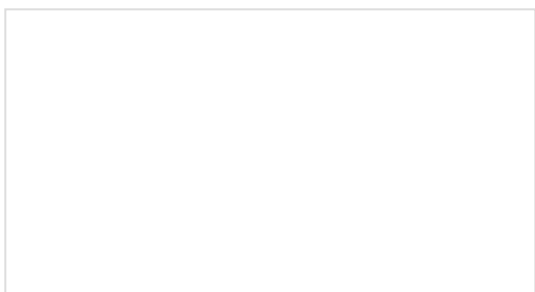
Raspberry Pi SPI and I2C Tutorial

Learn how to use serial I2C and SPI buses on your Raspberry Pi using the wiringPi I/O library for C/C++ and spidev/smbus for Python.



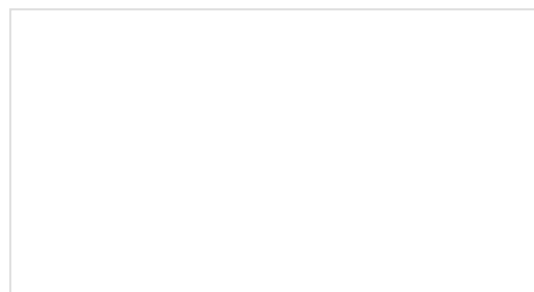
Python Programming Tutorial: Getting Started with the Raspberry Pi

This guide will show you how to write programs on your Raspberry Pi using Python to control hardware.



Qwiic pHAT for Raspberry Pi Hookup Guide

Get started interfacing your Qwiic enabled boards with your Raspberry Pi. This Qwiic connects the I2C bus (GND, 3.3V, SDA, and SCL) on your Raspberry Pi to an array of Qwiic connectors.



Qwiic Kit for Raspberry Pi Hookup Guide

Get started with the CCS811, BME280, VCNL4040, and microOLED via I2C using the Qwiic system and Python on a Raspberry Pi! Take sensor readings from the environment and display them on the microOLED, serial terminal, or the cloud with Cayenne!

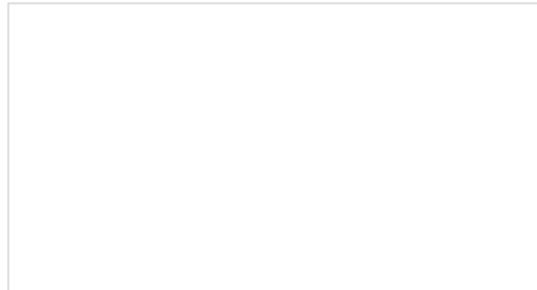




SparkFun BME280 Breakout Hookup Guide
A guide for connecting the BEM280 sensor to a microcontroller, and for using the Sparkfun Arduino library.



CCS811/BME280 (Qwiic) Environmental Combo Breakout Hookup Guide
Sense various environmental conditions such as temperature, humidity, barometric pressure, eCO2 and tVOCs with the CCS811 and BME280 combo breakout board.



RedBoard Qwiic Hookup Guide
This tutorial covers the basic functionality of the RedBoard Qwiic. This tutorial also covers how to get started blinking an LED and using the Qwiic system.



The Qwiic Atmospheric Sensor utilizes the Qwiic connect system. We recommend familiarizing yourself with the **Logic Levels** and **I²C** tutorials before using it. Click on the banner above to learn more about our Qwiic products.

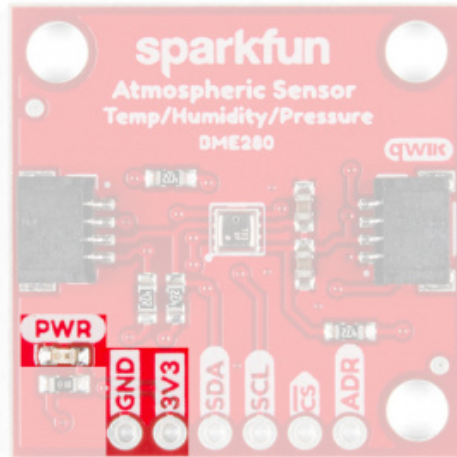
SparkFun's Qwiic Connect System



Hardware Overview

Power

There is a power status LED to help make sure that your Qwiic Atmospheric Sensor is getting power. You can power the board either through the *polarized Qwiic connector* system or the breakout pins (**3.3V** and **GND**) provided. This Qwiic system is meant to use **3.3V**, be sure that you are **NOT** using another voltage when using the Qwiic system.

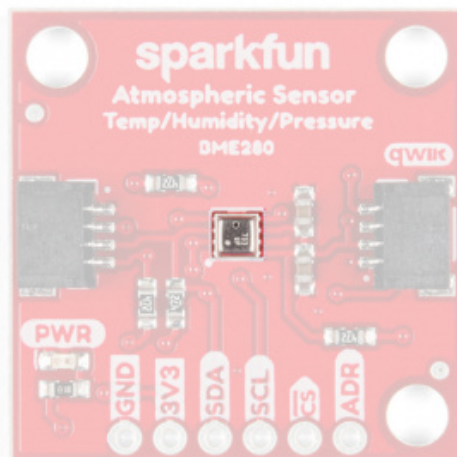


Annotated image of power LED along with VCC and GND connections. (Click to enlarge)

If you want to conserve power, there is an available jumper on the back of the board labeled LED to cut power to the LED (see **Jumpers** section below).

BME280 Sensor

The Bosch BME280 is the atmospheric sensor used on this board. It measures three different atmospheric properties: ambient temperature, (relative) humidity, and barometric pressure.



Annotated image of BME280 sesnor. (Click to enlarge)

The BME280 uses three modes of operation: sleep mode, forced mode and normal mode. These modes dictate how the sensor performs measurement cycles. The BME280 can be interfaced via I²C (*used with Qwiic connections*) or SPI communication. In the table below, are some of the characteristics of the BME280 sensor from the datasheet:

Characteristic	Description
Operating Voltage (V _{DD})	1.71V to 3.6V (Default on Qwiic System: 3.3V)
Operational Modes	**Sleep** (**Default**), Normal, and Forced (<i>low power; single measurement</i>)
Data Output	16-bit output from ADC (*IIR filter and oversampling can increase this to 20-bit; excludes humidity data.)
Current Consumption (Typical)	Sleep: 0.3 µA Standby: 0.5 µA (inactive period of normal mode) Humidity Measurements: 340 µA (peaks at 85°C) Pressure Measurements: 714 µA (peaks at -40°C) Temperature Measurements: 350 µA (peaks at 85°C)
Humidity Parameters	Range: 0 to 100 %RH Absolute Accuracy: ±3 %RH (from 20 - 80 %RH) Resolution: 0.008 %RH Forced Mode Current Consumption: 2.8 µA (max)
Pressure Parameters	Range: 300 to 1100 hPa (30,000 - 110,000 Pa or approx. 4.35 - 15.95 PSI) Absolute Accuracy: ±(1 - 1.7) hPa Resolution: 0.18 Pa Forced Mode Current Consumption: 4.2 µA (max)
Temperature Parameters	Range: 0°C to 65°C (32°F to 149°F) Absolute Accuracy: ±(0.5 - 1.5)°C Resolution: 0.01°C Forced Mode Current Consumption: 1.0 µA (typical)
I ² C Address	0x77 (Default) or 0x76

Modes of Operation

The BME280 offers three modes of operation:

- **Sleep mode (Default):** No operation, all registers accessible, lowest power, selected after startup.
- **Forced mode (*low power operation*):** Performs one measurement, store results and return to sleep mode.
- **Normal mode (*active measurements*):** Perpetual cycling of measurements and inactive periods.

For more details, refer to sections 3.3.2-4 in the datasheet. Additionally, in section 3.5 there are a set of recommended sensor settings for various applications or operations.

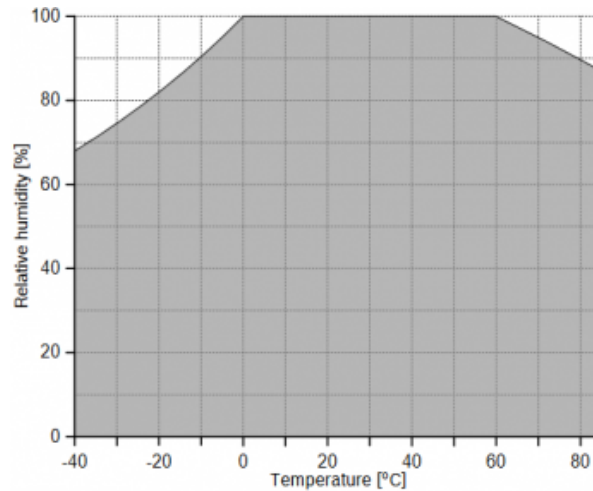
Measurements

All measurements can be skipped or enabled. When enabled, there are several oversampling options; with oversampling, it is possible to reduce the noise.

For more details, refer to sections 3-4 in the datasheet.

Humidity

The resolution of the humidity measurement is fixed at 16-bit ADC output. A graph of the operational range for the humidity sensor (*shaded in grey*) is shown below; the sensor will not report and/or operate properly outside of this range.



Operational range of the humidity sensor. (Click to enlarge)

Pressure and Temperature

For temperature and pressure readings, the resolution of the data will be dependent on if the infinite impulse response (IIR) filter is enabled and the oversampling setting register setting (osrs):

- When the IIR filter is enabled, the measurement resolution is 20-bit.
- When the IIR filter is disabled, the measurement resolution is $[16 + (\text{osrs}-1)]$ -bit.
 - e.g. The temperature measurement is 18-bit when `osrs_t` is set to '3'.

*(*Note: The temperature value depends on the PCB temperature, sensor element self-heating and ambient temperature and is typically just above ambient temperature.)*

Data Analysis

Below are other important attributes of the sensor. For most users, this information is will either be outside their scope or trivial. However for those that are interested, these topics have been briefly summarized or quoted directly from the datasheet. For full details, please refer to the datasheet; additionally, some of the comments in the library may help.

Infinite Impulse Response Filter

- It is recommended that the internal IIR filter be implemented to dampen rapid data fluctuations from external influences like wind blowing, closing doors, etc.

Noise

- The expected noise in the measurement data is dependent on the oversampling setting. For pressure and temperature readings, it is also dependent on the IIR filter setting used.

Trimming Parameters

- The trimming parameters are programmed into the devices' non-volatile memory (NVM) during production and cannot be altered by users. These are used for the calibration/compensation parameters.

Compensation formulas

- It is strongly advised (by the manufacturer) to use the API available from Bosch Sensortec to perform readout and compensation.

Qwiic or I²C

I²C Address

The BME280 has 2 available I²C addresses, which are set by the address pin, ADR. On the Qwiic Atmospheric sensor, the default slave address of the BME280 is **0x77** (HEX).

Default I²C Slave Address: 0x77

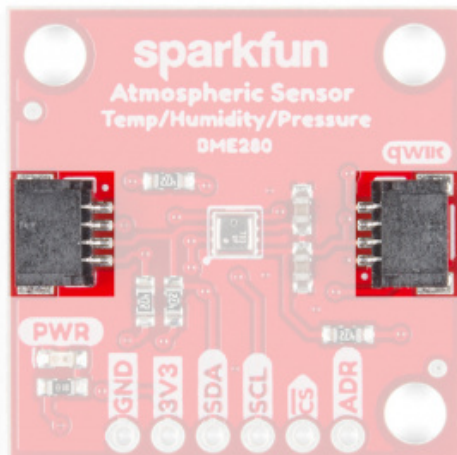
I²C Registers

The BME280 register (*memory*) map is detail in section 5.3 of the datasheet.

Address	Description
0xD0	ID: The chip identification number.
0xE0	Soft Reset: If the value 0xB6 is written to the register, the device is reset using the complete power-on-reset procedure
0xF2	ctrl_hum: Sets the humidity data acquisition options of the device. Changes to this register only become effective after a write operation to ctrl_meas .
0xF3	status: Indicate the status of the device. Whether a conversion is running or the results have been transferred to the data registers. Whether NVM data are being copied to image registers.
0xF4	ctrl_meas: Sets the pressure and temperature data acquisition options of the device. The register needs to be written after changing ctrl_hum for the changes to become effective./td>
0xF5	config: Sets the rate, filter and interface options of the device. Writes to the config register in <i>normal mode</i> may be ignored. In <i>sleep mode</i> writes are not ignored.
0xF7 to 0xF9	press: The raw pressure measurement data.
0xFA to 0xFC	temp: The raw temperature measurement data.
0xFD to 0xFE	hum: The raw humidity measurement data.
0xE1 to 0xF0 0x88 to 0xA1	Calibration Data: Holds <i>Trimming Parameters</i> .

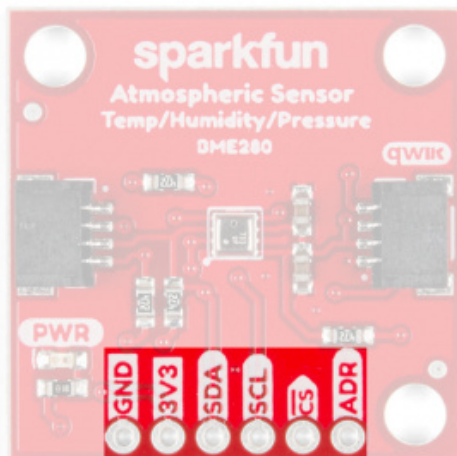
Connections

The simplest way to use the Qwiic ADC is through the Qwiic connect system. The connectors are polarized for the I²C connection and power. (**They are tied to their corresponding breakout pins.*)



Annotated image of the Qwiic connectors.

However, the board also provides six labeled breakout pins. You can connect these lines to the I²C bus of your microcontroller and power pins (**3.3V** and **GND**), if it doesn't have a Qwiic connector. Otherwise, the breakout pins can also be used for an SPI connection.



Annotated image of the breakout pins.

Pin Label	Pin Function	Input/Output	Notes
3.3V	Power Supply	Input	3.3V on Qwiic system (<i>should be stable</i>)
GND	Ground	Input	Ground and Single-Ended Reference Voltage for ADC.

SDA	I ² C Data Signal	Bi-directional	Bi-directional data line. Voltage should not exceed power supply (e.g. 3.3V).
SCL	I ² C Clock Signal	Input	Master-controlled clock signal. Voltage should not exceed power supply (e.g. 3.3V).

SPI Connection

There are two options for an SPI connection 3-wire or 4-wire. For a 3-wire connection, users will need to cut the `ADR` jumper on the board. For a 4-wire connection users can cut the `ADR`, `I2C`, and `CS` jumpers to remove the load from the SPI lines, but it is not necessary. For more details, check out the notes in the schematic.

This tutorial will not go into detail about using an SPI connection as the Python library can only be used with an I²C connection. However, for users seeking an SPI setup, they can refer to the hookup guide for the original BME280 Sensor Breakout board.

Pin Label	Pin Function	Input/Output	Notes
3.3V	Power Supply	Input	Supply voltage for sensor. Should be regulated between 1.8 and 3.6 V
GND	Ground	Input	Ground
SCK	Clock Signal	Input	Master-controlled clock signal. Voltage should not exceed power supply (max. 3.6V).
SDI	Data In	Input	Data incoming to the BME280. Voltage should not exceed power supply (max. 3.6V).
SDO	Data Out	Output	Data coming from the BME280.
CS	Chip Select (or <i>Slave Select</i>)	Input	Used to select device communication on 4-wire connections (active low). Voltage should not exceed power supply (max. 3.6V).

Jumpers

Caution: Be careful when cutting traces, as not to unintentionally cut other traces.

There are 4 separate jumpers on the board for various hardware related functions. For more notes, check out the hardware schematic. Not sure how to modify a jumper? [Read here!](#)

LED Power

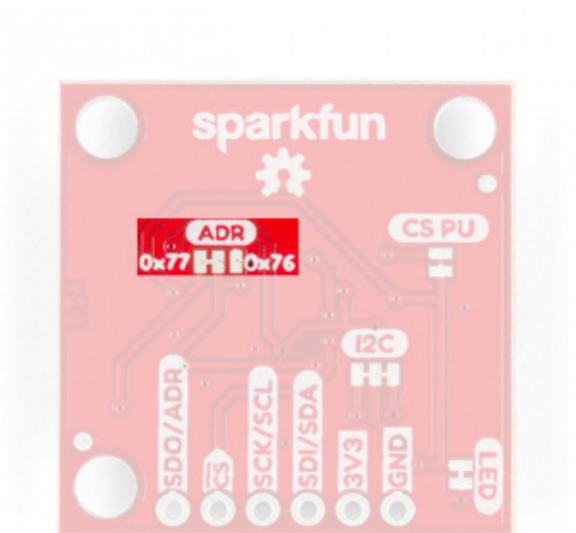
If you want to conserve power, the jumper labeled LED will allow users to isolate power to the power status indicator LED.



Annotated image of LED jumper. (Click to enlarge)

I²C Address

If you want to change the I²C address for the sensor, the jumper labeled ADR will allow users to change the I²C address from the default (**0x77**) to **0x76**.



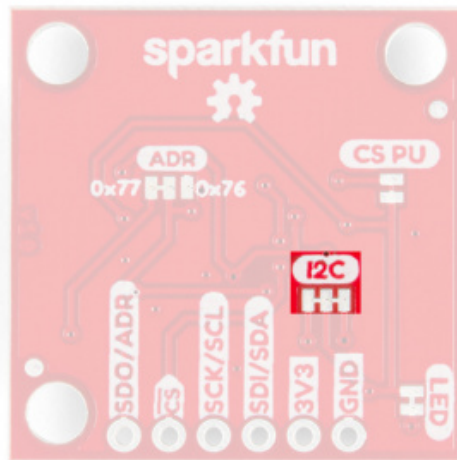
Annotated image of ADR jumper. (Click to enlarge)

Pull-Up Resistors

There are two jumpers (well technically three) for the pull-up resistors attached to specific pins on the sensor.

I²C Pull-Ups

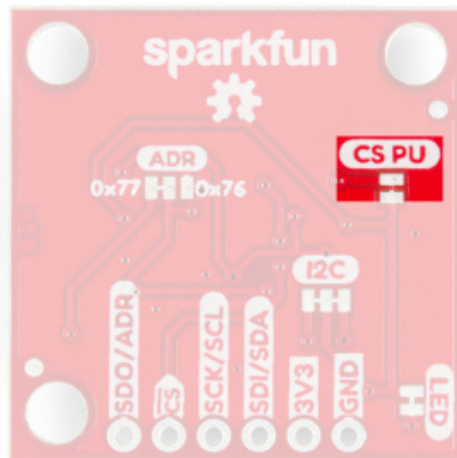
The first set of pull-up resistors are tied to the SDA and SCL lines for an I²C connection. Cutting the I2C jumper will remove the **4.7kΩ** pull-up resistors from the I²C bus. If you have many devices on your I²C bus you may want to remove these jumpers. (*When there are multiple devices on the bus with pull-up resistors, the equivalent parallel resistance may create too strong of a pull-up for the bus to operate correctly.*)



Annotated image of I2C jumper. (Click to enlarge)

CS (and SPI) Pull-Ups

The last pull-up resistor is tied to CS pin for an SPI connection. Cutting the CS jumper will remove the **4.7kΩ** pull-up resistor.



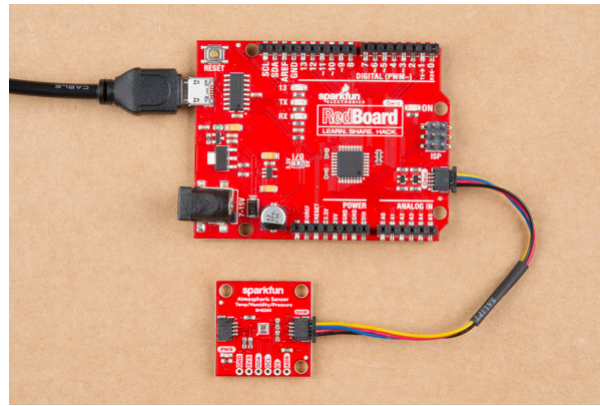
Annotated image of I2C jumper. (Click to enlarge)

Keep in mind that the rest of the SPI pins are shared with other pins (see *the note above on the SPI connection or the schematic*). For a 3-wire connection, users will need to cut the ADR jumper for the SDO line. For a 4-wire connection users can cut the ADR, I2C, and CS jumpers to remove the load from the SPI lines, but it is not necessary.

Hardware Assembly

Arduino Examples

With the Qwiic connector system, assembling the hardware is simple. All you need to do is connect your Qwiic Atmospheric Sensor (BME280) to the RedBoard Qwiic with a Qwiic cable. Otherwise, you can use the I²C pins of your microcontroller; just be aware of logic levels.



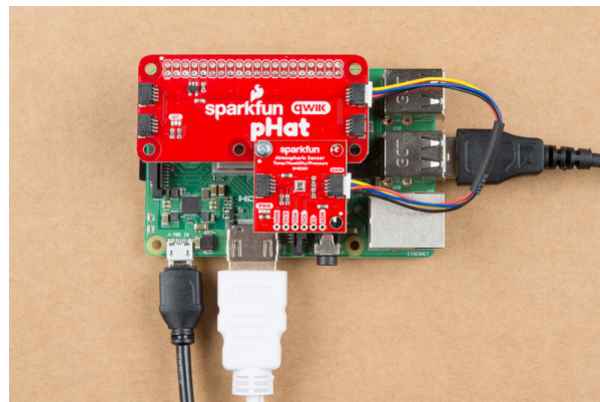
RedBoard Qwiic connected the Qwiic Atmospheric Sensor with a Qwiic cable.

Note: This tutorial assumes you are familiar with Arduino products and you are using the latest stable version of the Arduino IDE on your desktop. If this is your first time using the Arduino IDE, please review our tutorial on installing the Arduino IDE.

Raspberry Pi Example

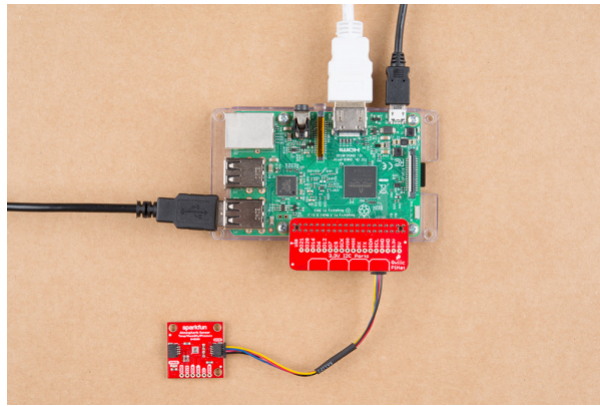
Note: This sensor and the Python library have not been tested on the newly released Raspberry Pi 4 because we don't carry it in our catalog yet.

With the Qwiic connector system, assembling the hardware is simple. In addition to the Qwiic Atmospheric Sensor (BME280), you will need: a Qwiic cable, a SparkFun Qwiic pHAT for Raspberry Pi, and a Raspberry Pi setup with the Raspbian OS, monitor, and standard peripherals. (**If you are unfamiliar with the Qwiic pHAT, you can find the Hookup Guide here.*)



*Raspberry Pi 3B+ connected the Qwiic Atmospheric Sensor with a Qwiic pHAT and Qwiic cable.
(*A 4-40 screw and nut were used to mount the sensor to the pHAT.)*

Alternatively, you can also use a Raspberry Pi 3 and the Qwiic HAT instead.



Raspberry Pi 3 connected the Qwiic Atmospheric Sensor with a Qwiic HAT and Qwiic cable.

Note: This tutorial assumes you are familiar with using a Raspberry Pi and you have the latest (full... with recommended software) version of Raspbian OS your Raspberry Pi. You can download the latest version of the Raspbian OS from the Raspberry Pi Foundation website. As of Feb. 13th 2019, we recommend the **Raspbian Stretch with desktop and recommended software** option.

If this is your first time using a Raspberry Pi, please head over to the Raspberry Pi Foundation website to use their quickstart guides. We have listed a few of them here:

1. Setting up your Raspberry Pi
2. Using your Raspberry Pi
3. Documentation:
 - Setup Documentation
 - Installation Documentation
 - Raspbian Documentation
 - SD card Documentation

Arduino Library Overview

Note: This example assumes you are using the latest version of the Arduino IDE on your desktop. If this is your first time using Arduino, please review our tutorial on installing the Arduino IDE. If you have not previously installed an Arduino library, please check out our installation guide.

We've written a library to easily get setup and take readings from the Qwiic Atmospheric Sensor. However, before we jump into getting data from the sensor, let's take a closer look at the available functions in the library. You can install this library through the Arduino Library Manager. Search for **SparkFun BME280 Arduino Library** and you should be able to install the latest version. If you prefer manually downloading the libraries from the GitHub repository, you can grab them here:

[DOWNLOAD THE SPARKFUN BME280 LIBRARY \(ZIP\)](#)

Let's get started by looking at the functions that set up the Qwiic Atmospheric Sensor:

Class

In the global scope, construct your sensor object (such as `mySensor` or `pressureSensorA`) without arguments.

BME280 mySensor;

Object Parameters and setup()

Rather than passing a bunch of data to the constructor, configuration is accomplished by setting the values of the BME280 type in the `setup()` function. They are exposed by being `public`: so use the `myName.aVariable = someValue;` syntax.

Settable variables of the class BME280:

```
//Main Interface and mode settings
uint8_t commInterface;
uint8_t I2CAddress;
uint8_t chipSelectPin;

uint8_t runMode;
uint8_t tStandby;
uint8_t filter;
uint8_t tempOverSample;
uint8_t pressOverSample;
uint8_t humidOverSample;
```

Functions

.begin();

Initialize the operation of the BME280 module with the following steps:

- Starts up the wiring library for I²C by default
- Checks/Validates BME280 chip ID
- Reads compensation data
- Sets default settings from table
- Sets operational mode to *Normal Mode*

Output: `uint8_t`

Returns the BME280 chip ID stored in the ID register.

.begin() Needs to be run once during the setup, or after any settings have been modified. In order to let the sensor's configuration take place, the BME280 requires a minimum time of about 2 ms in the sketch before you take data.

.beginSPI(uint8_t csPin);

Begins communication with the BME280 over an SPI connection.

Input: `uint8_t`

csPin: Digital pin used for the CS.

Output: Boolean

True: Connected to sensor.

False: Unable to establish connection.

.beginI2C(TwoWire &wirePort); or **.beginI2C(SoftwareWire &wirePort);**

Begins communication with the BME280 over an I²C connection. If `#ifdef SoftwareWire_h` is defined, then a software I²C connection is used.

Input: &wirePort

&wirePort: Port for the I²C connection.

Output: Boolean

True: Connected to sensor.

False: Unable to establish connection.

`.setMode(uint8_t mode);`

Sets the operational mode of the sensor. *(For more details, see section 3.3 of the datasheet.)*

Input: uint8_t

0: Sleep Mode

1: Forced Mode

2: Normal Mode

`.getMode();`

Returns the operational mode of the sensor.

Output: uint8_t

0: Sleep Mode

1: Forced Mode

2: Normal Mode

`.setStandbyTime(uint8_t timeSetting);`

Sets the standby time of the cycle time. *(For more details, see section 3.3 and Table 27 of the datasheet.)*

Input: uint8_t

0: 0.5ms

1: 62.5ms

2: 125ms

3: 250ms

4: 500ms

5: 1000ms

6: 10ms

7: 20ms

`.setFilter(uint8_t filterSetting)`

Sets the time constant of the IIR filter, which slows down the response time of the sensor inputs based on the number of samples required. *(For more details, see section 3.4.4, Table 6, and Figure 7 of the datasheet.)*

Input: uint8_t

0: filter off

1: coefficient of 2

2: coefficient of 4

3: coefficient of 8

4: coefficient of 16

`.setTempOverSample(uint8_t overSampleAmount);`

Sets the oversampling option (`osrs_t`) for the temperature measurements. *(Directly influences the noise and resolution of the data.)*

Input: uint8_t

0: turns off temperature sensing
1: oversampling ×1
2: oversampling ×2
4: oversampling ×4
8: oversampling ×8
16: oversampling ×16
Other: Bad Entry, sets to *oversampling ×1* by default.

Note: Yes, we do know there is a spelling error in the name of the method. It will get corrected in the next library update.

`.setPressureOverSample(uint8_t overSampleAmount);`

Sets the oversampling option (`osrs_p`) for the pressure measurements. (*Directly influences the noise and resolution of the data.*)

Input: `uint8_t`

0: turns off pressure sensing
1: oversampling ×1
2: oversampling ×2
4: oversampling ×4
8: oversampling ×8
16: oversampling ×16
Other: Bad Entry, sets to *oversampling ×1* by default.

`.setHumidityOverSample(uint8_t overSampleAmount);`

Sets the oversampling option (`osrs_h`) for the humidity measurements. (*Directly influences the noise of the data.*)

Input: `uint8_t`

0: turns off humidity sensing
1: oversampling ×1
2: oversampling ×2
4: oversampling ×4
8: oversampling ×8
16: oversampling ×16
Other: Bad Entry, sets to *oversampling ×1* by default.

`.setI2CAddress(uint8_t address);`

Changes the I²C address stored in the library to access the sensor.

Input: `uint8_t`

address: The new I²C address.

`.isMeasuring();`

Checks the `measuring` bit of the `status` register for if the device is taking measurement.

Output: Boolean

True: A conversion is running.

False: The results have been transferred to the data registers.

`.reset();`

Soft resets the sensor. (*If called, the `begin` function must be called before using the sensor again.*)

`.readFloatPressure();`

Reads raw pressure data stored in register and applies output compensation (*For more details on the data compensation, see section 4.2 of the datasheet.*)

Output: float

Returns pressure in kPa.

`.readFloatHumidity();`

Reads raw humidity data stored in register and applies output compensation (*For more details on the data compensation, see section 4.2 of the datasheet.*)

Output: float

Returns humidity in %RH.

`.readTempC();`

Reads raw temperature data stored in register and applies output compensation (*For more details on the data compensation, see section 4.2 of the datasheet.*)

Output: float

Returns temperature in Celsius.

`.readTempF();`

Reads raw temperature data stored in register and applies output compensation (*For more details on the data compensation, see section 4.2 of the datasheet.*)

Output: float

Returns temperature in Fahrenheit.

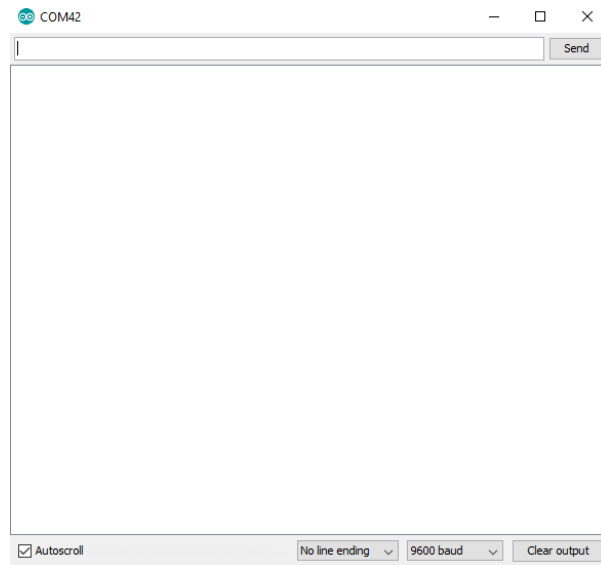
Arduino Example Code

The examples can be found in the the drop-down menu (**File > Examples > SparkFun BME280 > ...**) of the Arduino IDE. They can also be downloaded from the GitHub repository and then ran on their own. These are just a few samples of the available selection of examples.

Below are a sample readouts from the Serial Monitor for each of the examples. The baud rate for all of the examples default to **9600 baud**.

Example 1: Basic Readings

This basic example configures an BME280 on the I2C bus and reports out the data.

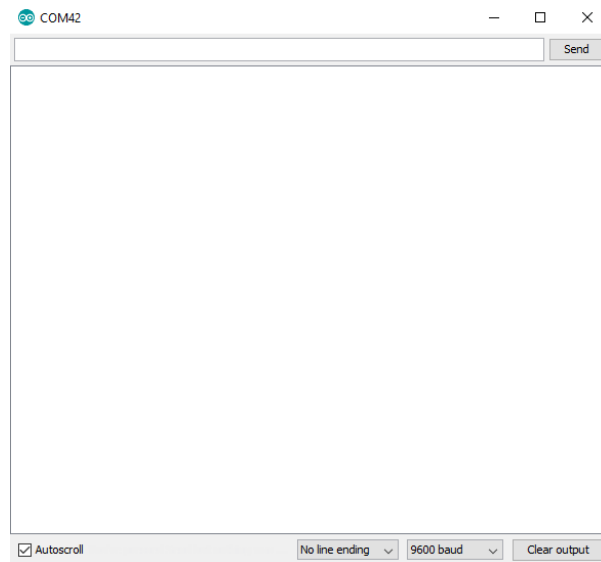


Expected readout from Example 1, where the basic readings are displayed

Example 3: CSV Output

If you want to use the BME280 to record data as a function of time, this example is for you! It outputs text as CSV (comma separated vales) that can be copy-pasted into a textfile or spreadsheet app for graphing.

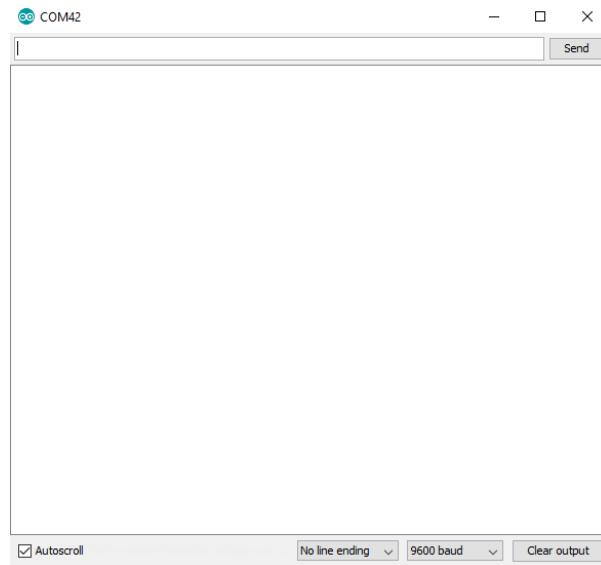
A note on accuracy: This sketch use "delay(50);" to wait 50ms between reads. The units of the 'sample' column are in (50ms + time-to-read) periods.



Expected readout from Example 3, where the first few lines show the generated CSV.

Example 5: Read All Registers

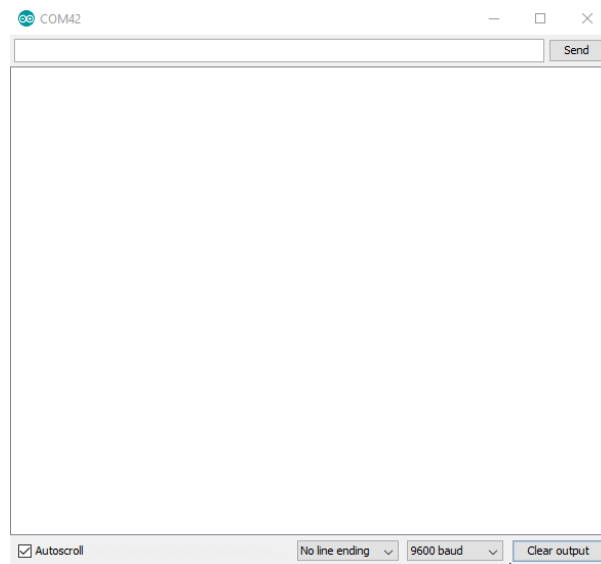
Here's an example that prints out the registers as well as the internally concatenated calibration words. It can be used to check the state of the BME280 after a particular configuration or can be implanted in your own sketch where you need to debug.



Expected readout from Example 5, where the full contents of memory are shown and a calculated pressure altitude.

Example 6: Low Power

Here is an example with a low power operation, utilizing the sleep mode functionality. This is similar to Example 1, except the sensor is in sleep mode and data is taken at specific intervals.



Expected readout from Example 6.

Python Library Overview

Note: This sensor and the Python library have not been tested on the newly released Raspberry Pi 4 because we don't carry it in our catalog yet.

Note: This example assumes you are using the latest version of Python (2 or 3). If this is your first time using Python or I²C hardware on a Raspberry Pi, please check out our tutorial on Python Programming with the Raspberry Pi and the Raspberry Pi SPI and I2C Tutorial.

We've written a library to easily get setup and take readings from the Qwiic Atmospheric Sensor. However, before we jump into getting data from the sensor, let's take a closer look at the available functions in the library. You can install the `sparkfun-qwiic-bme280` library package hosted by PyPi. However, if you prefer to manually download and build the libraries from the GitHub repository, you can grab them here (**Please be aware of any package dependencies. You can also check out the repository documentation page, hosted on Read the Docs.**):

DOWNLOAD THE SPARKFUN BME280 LIBRARY (ZIP)

Installation

Note: Don't forget to double check that the hardware I²C connection is enabled on your Raspberry Pi or other single board computer.

PyPi Installation

This repository is hosted on PyPi as the `sparkfun-qwiic-bme280` package . On systems that support PyPi installation via `pip` (use `pip3` for Python 3) is simple, using the following commands:

For **all users** (note: the user must have **sudo** privileges):

```
sudo pip install sparkfun-qwiic-bme280
```

For the **current user**:

```
pip install sparkfun-qwiic-bme280
```

Local Installation

To install, make sure the `setuptools` package is installed on the system.

Direct installation at the command line:

```
python setup.py install
```

To build a package for use with pip:

```
python setup.py sdist
```

A package file is built and placed in a subdirectory called `dist`. This package file can be installed using pip.

```
cd dist
pip install sparkfun_qwiic_bme280-<version>.tar.gz
```

Library Operation

Below is a description of the basic functionality of the Python library. This includes the library organization, built-in methods, and their inputs and/or outputs. For more details on how the library works, check out the source code and the sensor datasheet.

Dependencies

This library has a very few dependencies in the code, listed below:

```
from __future__ import print_function
import math
import qwiic_i2c
```

Default Variables

The default variables, in the code, for this library are listed below:

```
#The name of this device
_DEFAULT_NAME = "Qwiic BME280"

_AVAILABLE_I2C_ADDRESS = [0x77, 0x76]

#Default Setting Values
_settings = {"runMode" : 3,          \
            "tStandby" : 0,         \
            "filter"   : 0,         \
            "tempOverSample" : 1,   \
            "pressOverSample" : 1,  \
            "humidOverSample" : 1,  \
            "tempCorrection" : 0.0}

#define our valid chip IDs
_validChipIDs = [0x58, 0x60]
```

Class

QwiicBme280() or **QwiicBme280(i2caddr)**

This library operates as a class object, allowing new instances of that type to be made. An `__init__()` constructor is used that creates a connection to an I²C device over the I²C bus using the default or specified I²C address.

The Constructor

A constructor is a special kind of method used to initialize (assign values to) the data members needed by the object when it is created.

`__init__(address=None, i2c_driver=None):`

Input: value

The value of the device address. If not defined, the library will use the default I²C address (**0x77**) stored under `_AVAILABLE_I2C_ADDRESS` variable. The other available address is **0x76** (set by the jumper on the bottom side of the board).

Input: *i2c_driver*

Loads the specified I²C driver; by default the Qwiic I²C driver is used: `qwiic_i2c.getI2CDriver()`. Users should use the default I²C driver and leave this field blank.

Output: Boolean

True: Connected to I²C device on the default (or specified) address.

False: No device found or connected.

Functions

A function that is an attribute of the class, which defines a method for instances of that class. In simple terms, they are objects for the operations (or methods) of the class.

`.is_connected()`

Determines if the BME280 device is connected to the system.

Output: Boolean

True: Connected to I²C device on the default (or specified) address.

False: No device found or connected.

`.begin()`

Initialize the operation of the BME280 module with the following steps:

- Checks/Validates BME280 chip ID
- Reads compensation data
- Sets default settings from table
- Sets operational mode to *Normal Mode*

Output: Boolean

True: The initialization was successful.

False: Invalid chip ID.

`.set_mode(mode)`

Sets the operational mode of the sensor. `.mode` is also a property that can be set with the instance variables:

`MODE_SLEEP` , `MODE_FORCED` , or `MODE_NORMAL` . (*For more details, see section 3.3 of the datasheet.*)

Input: value

0: Sleep Mode

1: Forced Mode

2: Normal Mode

`.get_mode()`

Returns the operational mode of the sensor.

Output: integer

0: Sleep Mode

1: Forced Mode

2: Normal Mode

`.set_standby_time(timeSetting)`

Sets the standby time of the cycle time. (*For more details, see section 3.3 and Table 27 of the datasheet.*)

Input: value

0: 0.5ms

1: 62.5ms

2: 125ms

3: 250ms

4: 500ms

5: 1000ms

6: 10ms

7: 20ms

`.set_filter(filterSetting)`

Sets the time constant of the IIR filter, which slows down the response time of the sensor inputs based on the number of samples required. (*For more details, see section 3.4.4, Table 6, and Figure 7 of the datasheet.*)

Input: value

- 0:** filter off
- 1:** coefficient of 2
- 2:** coefficient of 4
- 3:** coefficient of 8
- 4:** coefficient of 16

.set_temperature_oversample(overSampleAmount)

Sets the oversampling option (`osrs_t`) for the temperature measurements. (*Directly influences the noise and resolution of the data.*)

Input: value

- 0:** turns off temperature sensing
- 1:** oversampling ×1
- 2:** oversampling ×2
- 4:** oversampling ×4
- 8:** oversampling ×8
- 16:** oversampling ×16
- Other:** Bad Entry, sets to *oversampling ×1* by default.

Note: Yes, we do know there is a spelling error in the name of the method. It will get corrected in the next library update.

.set_pressure_oversample(overSampleAmount)

Sets the oversampling option (`osrs_p`) for the pressure measurements. (*Directly influences the noise and resolution of the data.*)

Input: value

- 0:** turns off pressure sensing
- 1:** oversampling ×1
- 2:** oversampling ×2
- 4:** oversampling ×4
- 8:** oversampling ×8
- 16:** oversampling ×16
- Other:** Bad Entry, sets to *oversampling ×1* by default.

.set_humidity_oversample(overSampleAmount)

Sets the oversampling option (`osrs_h`) for the humidity measurements. (*Directly influences the noise of the data.*)

Input: value

- 0:** turns off humidity sensing
- 1:** oversampling ×1
- 2:** oversampling ×2
- 4:** oversampling ×4
- 8:** oversampling ×8
- 16:** oversampling ×16
- Other:** Bad Entry, sets to *oversampling ×1* by default.

.is_measuring()

Checks the `measuring` bit of the `status` register for if the device is taking measurement.

Output: Boolean

True: A conversion is running.

False: The results have been transferred to the data registers.

`.reset()`

Soft resets the sensor. *(If called, the begin method must be called before using the sensor again.)*

`.read_pressure()`

Reads raw pressure data stored in register and applies output compensation *(For more details on the data compensation, see section 4.2 of the datasheet.)*

Output: float

Returns pressure in Pa.

`.read_humidity()`

Reads raw humidity data stored in register and applies output compensation *(For more details on the data compensation, see section 4.2 of the datasheet.)*

Output: float

Returns humidity in %RH.

`.get_temperature_celsius()`

Reads raw temperature data stored in register and applies output compensation *(For more details on the data compensation, see section 4.2 of the datasheet.)*

Output: float

Returns temperature in Celsius.

`.get_temperature_fahrenheit()`

Reads raw temperature data stored in register and applies output compensation *(For more details on the data compensation, see section 4.2 of the datasheet.)*

Output: float

Returns temperature in Fahrenheit.

Python Example Code

The following examples are available in the GitHub repository. To run the examples, simply download or copy the code into a file. Then, open/save the example file (if needed) and execute the code in your favorite Python IDE.

For example, with the default Python IDLE click **Run > Run Module** or use the `F5` key. To terminate the example use the `Ctrl + C` key combination.

Note: Yes, there is an `altitude_feet` function in the library that is available. However, that content is omitted from the Library Overview section due to the amount of confusion that was generated with the previous hookup guide. Some users were misinterpreting the reported "*altitude*" from the function as an exact value; when it actually is **calculated** from the (barometric) pressure measurement to report the equivalent pressure altitude based on an atmospheric model. If you want more details on this subject, look into how an altimeter works.

Example 1

This example uses the default configuration settings for the sensor. The temperature (°F), humidity (%RH), pressure (Pa), and *calculated* pressure altitude (ft.) are reported repeatedly.

```

#!/usr/bin/env python
#-----
# qwiic_env_bme280_ex1.py
#
# Simple Example for the Qwiic BME280 Device
#-----
#
# Written by SparkFun Electronics, May 2019
#
# This python library supports the SparkFun Electronics qwiic
# qwiic sensor/board ecosystem on a Raspberry Pi (and compatible) single
# board computers.
#
# More information on qwiic is at https://www.sparkfun.com/qwiic
#
# Do you like this library? Help support SparkFun. Buy a board!
#
#=====
# Copyright (c) 2019 SparkFun Electronics
#
# Permission is hereby granted, free of charge, to any person obtaining a copy
# of this software and associated documentation files (the "Software"), to deal
# in the Software without restriction, including without limitation the rights
# to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
# copies of the Software, and to permit persons to whom the Software is
# furnished to do so, subject to the following conditions:
#
# The above copyright notice and this permission notice shall be included in all
# copies or substantial portions of the Software.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
# IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
# FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
# AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
# LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
# OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
# SOFTWARE.
#=====
# Example 1
#
from __future__ import print_function
import qwiic_bme280
import time
import sys

def runExample():

    print("\nSparkFun BME280 Sensor Example 1\n")
    mySensor = qwiic_bme280.QwiicBme280()

    if mySensor.connected == False:
        print("The Qwiic BME280 device isn't connected to the system. Please check your connecti

```

```

on", \
        file=sys.stderr)
    return

mySensor.begin()

while True:
    print("Humidity:\t%.3f" % mySensor.humidity)

    print("Pressure:\t%.3f" % mySensor.pressure)

    print("Altitude:\t%.3f" % mySensor.altitude_feet)

    print("Temperature:\t%.2f" % mySensor.temperature_fahrenheit)

    print("")

    time.sleep(1)

if __name__ == '__main__':
    try:
        runExample()
    except (KeyboardInterrupt, SystemExit) as exErr:
        print("\nEnding Example 1")
sys.exit(0)

```

```

Python 3.5.3 Shell
File Edit Shell Debug Options Window Help
Python 3.5.3 (default, Sep 27 2018, 17:25:39)
[GCC 6.3.0 20170516] on linux
Type "copyright", "credits" or "license()" for more information.
>>>
Ln: 4 Col: 4

```

Expected readout from Example 1.

Experiment 4

This example configures the settings for the sensor before the temperature (°F), humidity (%RH), pressure (Pa), and *calculated* pressure altitude (ft.) are reported repeatedly.

```

#!/usr/bin/env python
#-----
# qwiic_env_bme280_ex4.py
#
# Simple Example for the Qwiic BME280 Device
#-----
#
# Written by SparkFun Electronics, May 2019
#
# This python library supports the SparkFun Electronics qwiic
# qwiic sensor/board ecosystem on a Raspberry Pi (and compatible) single
# board computers.
#
# More information on qwiic is at https://www.sparkfun.com/qwiic
#
# Do you like this library? Help support SparkFun. Buy a board!
#
#=====
# Copyright (c) 2019 SparkFun Electronics
#
# Permission is hereby granted, free of charge, to any person obtaining a copy
# of this software and associated documentation files (the "Software"), to deal
# in the Software without restriction, including without limitation the rights
# to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
# copies of the Software, and to permit persons to whom the Software is
# furnished to do so, subject to the following conditions:
#
# The above copyright notice and this permission notice shall be included in all
# copies or substantial portions of the Software.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
# IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
# FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
# AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
# LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
# OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
# SOFTWARE.
#=====
# Example 4 - port of the Arduino example 4
#

from __future__ import print_function
import qwiic_bme280
import time
import sys

def runExample():

    print("\nSparkFun BME280 Sensor Example 4\n")
    mySensor = qwiic_bme280.QwiicBme280()

    if mySensor.connected == False:
        print("The Qwiic BME280 device isn't connected to the system. Please check your connecti
on" \

```

```

    file=sys.stderr)
    return

mySensor.begin()

# setup the sensor
mySensor.filter = 1          # 0 to 4 is valid. Filter coefficient. See 3.4.4
mySensor.standby_time = 0    # 0 to 7 valid. Time between readings. See table 27.

mySensor.over_sample = 1     # 0 to 16 are valid. 0 disables temp sensing. See table
24.
mySensor.pressure_oversample = 1 # 0 to 16 are valid. 0 disables pressure sensing. See ta
ble 23.
mySensor.humidity_oversample = 1 # 0 to 16 are valid. 0 disables humidity sensing. See ta
ble 19.
mySensor.mode = mySensor.MODE_NORMAL # MODE_SLEEP, MODE_FORCED, MODE_NORMAL is valid. See 3.
3

while True:
    print("Humidity:\t%.3f" % mySensor.humidity)

    print("Pressure:\t%.3f" % mySensor.pressure)

    print("Altitude:\t%.3f" % mySensor.altitude_feet)

    print("Temperature:\t%.2f\n" % mySensor.temperature_fahrenheit)

    time.sleep(1)

if __name__ == '__main__':
    try:
        runExample()

    except (KeyboardInterrupt, SystemExit) as exErr:
        print("\nEnding Example 4")
sys.exit(0)

```

```
Python 3.5.3 Shell
File Edit Shell Debug Options Window Help
Python 3.5.3 (default, Sep 27 2018, 17:25:39)
[GCC 6.3.0 20170516] on linux
Type "copyright", "credits" or "license()" for more information.
>>>
```

Expected readout from Example 4.

Experiment 5

This last example reports configuration the settings and calibration values from the memory map. Then, the temperature (°F), humidity (%RH), pressure (Pa), and *calculated* pressure altitude (ft.) are reported repeatedly.


```

#!/usr/bin/env python
#-----
# qwiic_env_bme280_ex5.py
#
# Simple Example for the Qwiic BME280 Device
#-----
#
# Written by SparkFun Electronics, May 2019
#
# This python library supports the SparkFun Electronics qwiic
# qwiic sensor/board ecosystem on a Raspberry Pi (and compatible) single
# board computers.
#
# More information on qwiic is at https:# www.sparkfun.com/qwiic
#
# Do you like this library? Help support SparkFun. Buy a board!
#
#=====
# Copyright (c) 2019 SparkFun Electronics
#
# Permission is hereby granted, free of charge, to any person obtaining a copy
# of this software and associated documentation files (the "Software"), to deal
# in the Software without restriction, including without limitation the rights
# to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
# copies of the Software, and to permit persons to whom the Software is
# furnished to do so, subject to the following conditions:
#
# The above copyright notice and this permission notice shall be included in all
# copies or substantial portions of the Software.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
# IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
# FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
# AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
# LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
# OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
# SOFTWARE.
#=====
# Example 5 - port of the Arduino example 5
#

from __future__ import print_function
import qwiic_bme280
import time
import sys

def runExample():

    print("\nSparkFun BME280 Sensor Example 5\n")
    mySensor = qwiic_bme280.QwiicBme280()

    if mySensor.connected == False:
        print("The Qwiic BME280 device isn't connected to the system. Please check your connecti

```

```

on", \
        file=sys.stderr)
    return

    mySensor.begin()

    print("ID(0xD0): 0x%.2x" % mySensor._i2c.readByte(mySensor.address, mySensor.BME280_CHIP_ID_
REG))
    print("Reset register(0xE0): 0x%.2x" % mySensor._i2c.readByte(mySensor.address, mySensor.BME
280_RST_REG))
    print("ctrl_meas(0xF4): 0x%.2x" % mySensor._i2c.readByte(mySensor.address, mySensor.BME280_C
TRL_MEAS_REG))
    print("ctrl_hum(0xF2): 0x%.2x\n" % mySensor._i2c.readByte(mySensor.address, mySensor.BME280_
CTRL_HUMIDITY_REG))

    print("Displaying all regs:")
    memCounter = 0x80
    for row in range(8,16):
        print("0x%.2x 0:" % row, end='')
        for column in range(0,16):
            tempReadData = mySensor._i2c.readByte(mySensor.address, memCounter)
            print("0x%.2x " % tempReadData, end='')

            memCounter += 1
        print("")

    print("Displaying concatenated calibration words:")
    print("dig_T1, uint16: %d" % mySensor.calibration["dig_T1"])
    print("dig_T2, int16: %d" % mySensor.calibration["dig_T2"])
    print("dig_T3, int16: %d" % mySensor.calibration["dig_T3"])
    print("dig_P1, uint16: %d" % mySensor.calibration["dig_P1"])
    print("dig_P2, int16: %d" % mySensor.calibration["dig_P2"])
    print("dig_P3, int16: %d" % mySensor.calibration["dig_P3"])
    print("dig_P4, int16: %d" % mySensor.calibration["dig_P4"])
    print("dig_P5, int16: %d" % mySensor.calibration["dig_P5"])
    print("dig_P6, int16: %d" % mySensor.calibration["dig_P6"])
    print("dig_P7, int16: %d" % mySensor.calibration["dig_P7"])
    print("dig_P8, int16: %d" % mySensor.calibration["dig_P8"])
    print("dig_P9, int16: %d" % mySensor.calibration["dig_P9"])
    print("dig_H1, uint8: %d" % mySensor.calibration["dig_H1"])
    print("dig_H2, int16: %d" % mySensor.calibration["dig_H2"])
    print("dig_H3, uint8: %d" % mySensor.calibration["dig_H3"])
    print("dig_H4, int16: %d" % mySensor.calibration["dig_H5"])
    print("dig_H6, int8: %d" % mySensor.calibration["dig_H6"])
    while True:

        print("Humidity:\t%.3f" % mySensor.humidity)

        print("Pressure:\t%.3f" % mySensor.pressure)

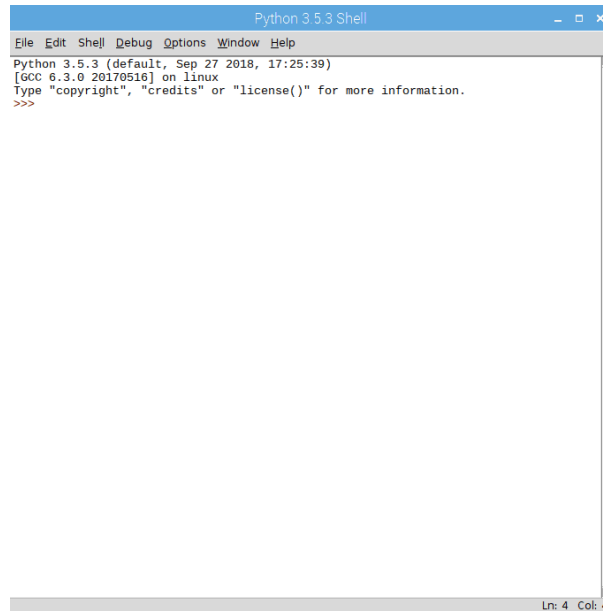
        print("Altitude:\t%.3f" % mySensor.altitude_feet)

        print("Temperature:\t%.2f\n" % mySensor.temperature_fahrenheit)

```

```
        time.sleep(1)

if __name__ == '__main__':
    try:
        runExample()
    except (KeyboardInterrupt, SystemExit) as exErr:
        print("\nEnding Example 5")
    sys.exit(0)
```

A screenshot of a Python 3.5.3 Shell window. The window title is "Python 3.5.3 Shell". The menu bar includes "File", "Edit", "Shell", "Debug", "Options", "Window", and "Help". The main content area shows the following text: "Python 3.5.3 (default, Sep 27 2018, 17:25:39)", "[GCC 6.3.0 20170516] on linux", "Type \"copyright\", \"credits\" or \"license()\" for more information.", and a prompt ">>>". The status bar at the bottom right indicates "Ln: 4 Col: 4".

Expected readout from Example 5.

Troubleshooting Tips

Here are a few tips for troubleshooting this device.

Power

If you are not using the Qwiic system, make sure your supply voltage is within the electrical specifications of the BME280.

No Available Devices

Double check your connections. On a Raspberry Pi, you may get this is indicated with an `OSError: [Errno 121] Remote I/O error` readout.

On a Raspberry Pi, also make sure that the I²C hardware is enabled. This is usually indicated with an `Error: Failed to connect to I2C bus 1.` readout.

SPI Setup

As mentioned earlier, this tutorial does not go into detail about using an SPI connection as the Python library can only be used with an I²C connection. However, for users seeking an SPI setup, they can refer to the hookup guide for the original BME280 Sensor Breakout board.

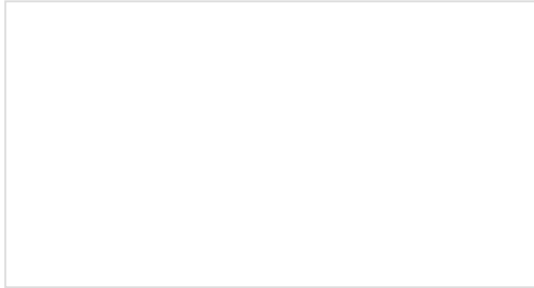
If you still have questions or issues with this product, please create a post on our forum under the Qwiic Atmospheric Sensor (SEN-15440): Questions and Issues topic.

Resources and Going Further

For more product information, check out the resources below:

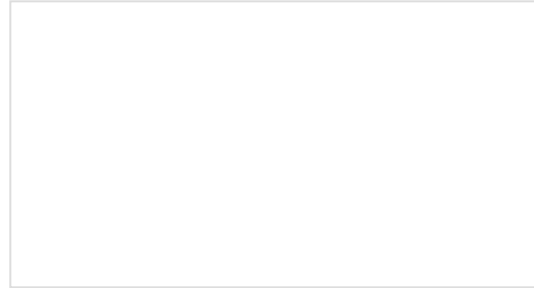
- Schematic (PDF)
- Eagle Files (ZIP)
- BME280 Datasheet
- SparkFun BME280 Arduino Library
- SparkFun BME280 Python Library
- GitHub Product Repo
- Product Video

Need some inspiration for your next project? Check out some of these other Qwiic product tutorials:



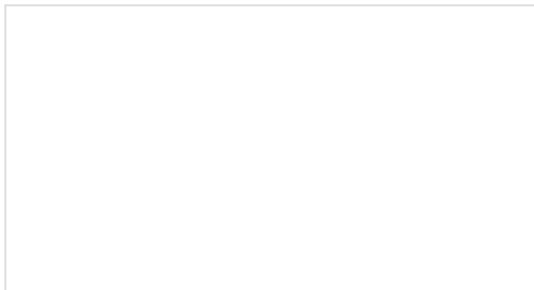
SparkFun LoRa Gateway 1-Channel Hookup Guide

How to setup and use the LoRa Gateway 1-Channel in Arduino.



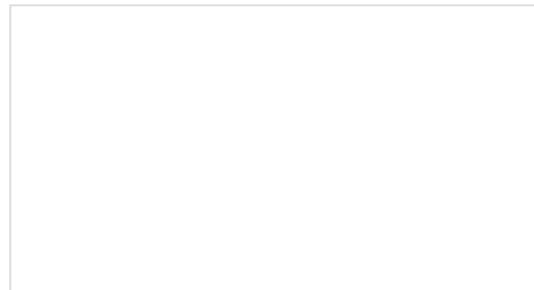
RedBoard Turbo Hookup Guide

An introduction to the RedBoard Turbo. Level up your Arduino-skills with the powerful SAMD21 ARM Cortex M0+ processor!



Qwiic Proximity Sensor (VCNL4040) Hookup Guide

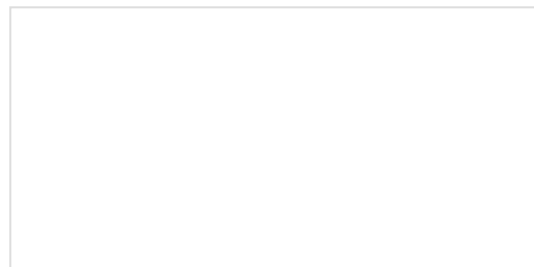
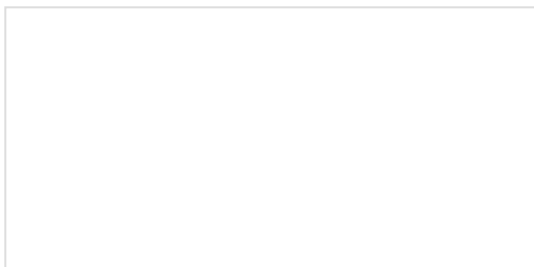
The SparkFun Qwiic Proximity Sensor is a great, qualitative proximity (up to 20 cm) and light sensor. This hookup guide covers a few examples to retrieve basic sensor readings.



SparkFun 9DoF IMU (ICM-20948) Breakout Hookup Guide

How to use the SparkFun 9DoF ICM-20948 breakout board for your motion sensing projects. This breakout is ideal for wearable sensors and IoT applications.

Need some inspiration for your next project? Check out some of these other weather related tutorials:

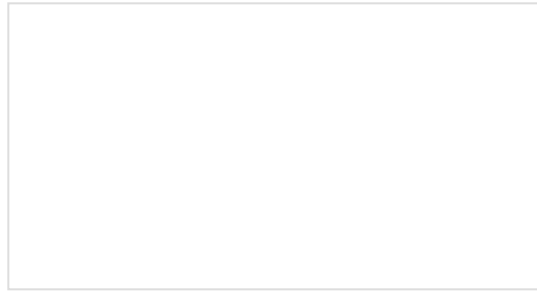


T5403 Barometric Pressure Sensor Hookup Guide

T5403 Barometric Pressure Sensor Hookup Guide

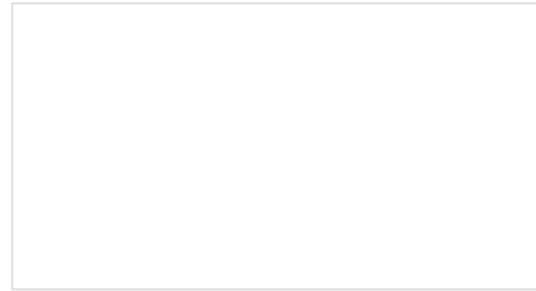
SparkFun Inventor's Kit for Photon Experiment Guide

Dive into the world of the Internet of Things with the SparkFun Inventor's Kit for Photon.



Photon Weather Shield Hookup Guide V11

Create Internet-connected weather projects with the SparkFun Weather Shield for the Photon.



Spectral Triad (AS7265x) Hookup Guide

Learn how to wield the power of 18 channels of UV to NIR spectroscopy with AS72651 (UV), AS72652 (VIS), and AS72653 (NIR) sensors!



Want more Python?

We are working on more tutorials, blogs, and product releases around the Python programming language.

Would you like to be notified when new content is available?

Email*

Would you also like to subscribe to SparkFun's weekly newsletter?

Yes, sign me up!