Table of Contents

**Overview**

- Choose Your Own Adventure 3
- Parts 4

**Assembly**

- Basic Wiring 5
- Using Bolt Kit 6

**Arduino Sender**

- Installing UF2 Examples 7
- Send CSV Formatted Text 7
- Send JSON Formatted Text 8
- Customizing 8
- Sensirion SCD4x Arduino Library 8
- Arduino Code 9

**CircuitPython Sender**

- Install CircuitPython 11
- Enable USB CDC Data 11
- CircuitPython Code 12

**Receiving Data**

- Install pySerial 13
- Which Serial Port? 13
- Read CSV String 14
- Read JSON String 14

**Direct Reading with U2IF**

- Install U2IF Firmware onto Trinkey 15
- Install Blinka onto PC 15
- Install Libraries onto PC 16
- Read Sensors via Python 16
Overview

Want to measure various air quality parameters, like humidity, temperature, CO2, etc.? Want to measure these directly into your computer? Want to do all this without soldering? No problem. In this guide we'll show how this can done using the Adafruit Trinkey QT2040 (https://adafru.it/Xfi).

We'll use a Sensirion SCD40 (https://adafru.it/XAH) for CO2 measurement. For temperature, pressure, and humidity, we'll use a Bosch BME280 (https://adafru.it/y8f). By connecting everything together using STEMMA QT cables (https://adafru.it/Tff) - no soldering is required.

Choose Your Own Adventure

This guide shows several different ways of coding up the Trinkey Enviro Sensor Gadget. Pick which ever adventure seems the most exciting! Or works best for your use case. Or try them all out!

Here is a summary:

- Arduino - Read via USB CDC serial. Precompiled UF2's are provided. Code changes require recompiling provided source with Arduino IDE.
- CircuitPython - Read via USB CDC serial. Code changes can be done directly to source without recompiling.
- U2IF - Read directly into PC using Python with Blinka. Code changes are done on the host PC.
Here is a list of the hardware items used in this project:

**Adafruit Trinkey QT2040 - RP2040 USB Key with Stemma QT**
It's half USB Key, half Adafruit QT Py, and a lotta RP2040...it's Trinkey QT2040, the circuit board with an RP2040 heart and Stemma QT legs....
https://www.adafruit.com/product/5056

**Adafruit SCD-40 - True CO2, Temperature and Humidity Sensor**
Take a deep breath in...now slowly breathe out. Mmm isn't it wonderful? All that air around us, which we bring into our lungs, extracts oxygen from and then breathes out carbon...
https://www.adafruit.com/product/5187

**Adafruit BME280 I2C or SPI Temperature Humidity Pressure Sensor**
Bosch has stepped up their game with their new BME280 sensor, an environmental sensor with temperature, barometric pressure and humidity! This sensor is great for all sorts...
https://www.adafruit.com/product/2652
Stacking M2.5 Hardware Kit for STEMMA QT and RP2040 Trinkey

Here is the perfect hardware kit to make a RP2040 QT Trinkey into any kind of USB-connected smart sensor with a...
https://www.adafruit.com/product/5248

STEMMA QT / Qwiic JST SH 4-Pin Cable

This 4-wire cable is 50mm / 1.9" long and fitted with JST SH female 4-pin connectors on both ends. Compared with the chunkier JST PH these are 1mm pitch instead of 2mm, but...
https://www.adafruit.com/product/4399

Assembly

Using STEMMA QT cables (https://adafru.it/Tff), plugging everything together is easy. And with the available bolt kit (https://adafru.it/Xfk), there are various ways in which the boards can be fastened together, or not, depending on how you want to arrange things.

Basic Wiring

Using STEMMA QT cables (https://adafru.it/Tff), the basic wiring is very simple - just plug everything together. There are several different STEMMA QT cable lengths to choose from. Also, the ordering of the sensors should not matter.
Using Bolt Kit

If you want to stack things together, the [M2.5 hardware kit](https://adafruit.it/Xfk) can be used. In order to stack more than two things together, you'll need more than one bolt kit. The SCD40 should end up on top to give it enough clearance.

Using two bolt kits, it's possible to stack all the boards together. However, this may be too chonky. The STEMMA QT cable may also interfere with plugging into USB port as well.

Using one bolt kit, just the BME280 and SCD40 can be stacked together. This allows the sensor package to be tethered separately from the Trinkey QT2040.

It is also OK to use no bolt kit at all and just connect the sensors in a free form chain.
Arduino Sender

The idea for the sending code is pretty simple - we just read the values in a loop and send them out over the serial port.

The complete Arduino sketch code is provided later to allow customizing. However, we provide some pre-compiled examples in UF2 format. With these UF2 files, you can just drag-and-drop to the Trinkey QT2040 RPI-RP2 bootloader folder. The Arduino IDE does not even need to be installed.

Installing UF2 Examples

To install the UF2 files:

1. Put the Trinkey QT2040 in bootloader mode by holding the BOOT button while pressing the RST (reset) button.
3. Drag the UF2 file to the RPI-RP2 folder.
4. Once copied, board should reset and code is now running.

Send CSV Formatted Text

This example provides comma separated (CSV) formatted text. Here's the UF2:

```
ENVIRO_CSV.uf2
```

[https://adafruit.it/XAI](https://adafruit.it/XAI)

With that running on the Trinkey QT2040, the serial output will look like this:
Send JSON Formatted Text

This example provides JavaScript Object Notation (JSON) formatted text. Here's the UF2:

![ENVIRO_JSON.uf2](https://adafru.it/XAJ)

With that running on the Trinkey QT2040, the serial output will look like this:

![Serial Output](image)

Customizing

You can customize the Arduino sketch sending behavior by modifying these lines found at the top of the sketch:

```cpp
//--| User Config |-----------------------------------------------
#define DATA_FORMAT   0         // 0=CSV, 1=JSON
#define DATA_RATE     5000      // generate new number ever X ms
#define BEAT_COLOR    0xADAF00  // neopixel heart beat color
#define BEAT_RATE     1000      // neopixel heart beat rate in ms, 0=none
//----------------------------------------------------------------
```

The ready to go UF2 examples provided above are just precompiled sketches with these already set for specific use cases, i.e. CSV and JSON output.

Sensirion SCD4x Arduino Library

To recompile the Arduino sketch, you'll need the Sensirion Arduino library for the SCD40:

![Sensirion SCD4x Arduino Library](https://adafru.it/Uxc)
It can be installed via the Arduino IDE Library Manager. Just search for "Sensirion I2C SCD4x":

![Library Manager SCD4x]  

Arduino Code

Here is the complete code listing for the Arduino sketch.

```cpp
// SPDX-FileCopyrightText: 2021 Carter Nelson for Adafruit Industries
// SPDX-License-Identifier: MIT

#include <Adafruit_NeoPixel.h>
#include <Adafruit_BME280.h>
#include <SensirionI2CScd4x.h>
#include <Wire.h>

//--| User Config |-----------------------------------------------
#define DATA_FORMAT 0         // 0=CSV, 1=JSON
#define DATA_RATE 5000      // generate new number ever X ms
#define BEAT_COLOR 0xADAF00  // neopixel heart beat color
#define BEAT_RATE 1000      // neopixel heart beat rate in ms, 0=none
//----------------------------------------------------------------
Adafruit_BME280 bme;
SensirionI2CScd4x scd4x;
Adafruit_NeoPixel neopixel(1, PIN_NEOPIXEL, NEO_GRB + NEO_KHZ800);

uint16_t CO2, data_ready;
float scd4x_temp, scd4x_humid;
float temperature, humidity, pressure;
int current_time, last_data, last_beat;

void setup() {
  Serial.begin(115200);

  // init status neopixel
  neopixel.begin();
  neopixel.fill(0);
  neopixel.show();

  // init BME280 first, this calls Wire.begin()
  if (!bme.begin()) {
    Serial.println("Failed to initialize BME280.");
    neoPanic();
  }
```

```cpp
```
// init SCD40
scd4x.begin(Wire);
scd4x.stopPeriodicMeasurement();
if (scd4x.startPeriodicMeasurement()) {
    Serial.println("Failed to start SCD40.");
    neoPanic();
}

// init time tracking
last_data = last_beat = millis();

void loop() {
current_time = millis();

// Send Data
//
if (current_time - last_data > DATA_RATE) {
    temperature = bme.readTemperature();
    pressure = bme.readPressure() / 100;
    humidity = bme.readHumidity();
    scd4x.setAmbientPressure(uint16_t(pressure));
    scd4x.readMeasurement(CO2, scd4x_temp, scd4x_humid);
    switch (DATA_FORMAT) {
        case 0:
            sendCSV(); break;
        case 1:
            sendJSON(); break;
        default:
            Serial.print("Unknown data format: "); Serial.println(DATA_FORMAT);
            neoPanic();
    }
    last_data = current_time;
}

// Heart Beat
//
if ((BEAT_RATE) && (current_time - last_beat > BEAT_RATE)) {
    if (neopixel.getPixelColor(0)) {
        neopixel.fill(0);
    } else {
        neopixel.fill(BEAT_COLOR);
    }
    neopixel.show();
    last_beat = current_time;
}

void sendCSV() {
    Serial.print(CO2); Serial.print(",");
    Serial.print(pressure); Serial.print(",");
    Serial.print(temperature); Serial.print(",");
    Serial.println(humidity);
}

void sendJSON() {
    Serial.print("\"CO2\""); Serial.print(CO2); Serial.print(",");
    Serial.print("\"pressure\""); Serial.print(pressure); Serial.print(",");
    Serial.print("\"temperature\""); Serial.print(temperature); Serial.print(",");
    Serial.print("\"humidity\""); Serial.print(humidity);
    Serial.println();
}

void neoPanic() {

}
CircuitPython Sender

This is basically the same idea as the Arduino sketch - just read the values and send over serial in a loop. There are CircuitPython libraries (https://adafru.it/ENC) for the SC D40 (https://adafru.it/XAK) and BME280 (https://adafru.it/BfX), which makes this easy to implement.

Install CircuitPython

This is covered in the Trinkey QT2040 main guide:

CircuitPython Quickstart
https://adafru.it/XAL

The required libraries will be included with the project bundle download along with project code below.

Enable USB CDC Data

We could probably just send the data using the typical serial output via the typical print() command. However, here we show how to enable and use the secondary USB CDC serial port available in CircuitPython:

USB Serial: Console (REPL) and Data
https://adafru.it/SC9

As mentioned in that guide, this requires a special boot.py file to be located in the CI RCUITPY folder. Well, not too special, it's just a few lines of code:

```python
import usb_cdc
usb_cdc.enable(data=True)
```

Copy that code and save it to CIRCUITPY/boot.py.
CircuitPython Code

Here is the project code. To get code and necessary libraries, click on the Download Project Bundle link below, and uncompress the .zip file. Once the sensors are connected and the modified boot.py has been put in place, simply save the project code as CIRCUITPY/code.py:

```python
# SPDX-FileCopyrightText: 2021 Carter Nelson for Adafruit Industries
#
# SPDX-License-Identifier: MIT

import time
import board
import usb_cdc
import adafruit_scd4x
from adafruit_bme280 import basic as adafruit_bme280
import neopixel

#--| User Config |-----------------------------------
DATA_FORMAT = "JSON"  # data format, CSV or JSON
DATA_RATE = 5         # data read rate in secs
BEAT_COLOR = 0xADAF00  # neopixel heart beat color
BEAT_RATE = 1         # neopixel heart beat rate in secs, 0=none
#----------------------------------------------------
# check that USB CDC data has been enabled
if usb_cdc.data is None:
    print("Need to enable USB CDC serial data in boot.py.")
    while True:
        pass

# setup stuff
scd = adafruit_scd4x.SCD4X(board.I2C())
scd.start_periodic_measurement()
bme = adafruit_bme280.Adafruit_BME280_I2C(board.I2C())
pixel = neopixel.NeoPixel(board.NEOPIXEL, 1)

# CSV output
def send_csv_data(values):
    usb_cdc.data.write("{}, {}, {}, {}
        .format(*values).encode())

# JSON output
def send_json_data(values):
    usb_cdc.data.write('{}
        .format(*values[0]).encode())
    usb_cdc.data.write('"pressure" : {},
        .format(values[1]).encode())
    usb_cdc.data.write('"temperature" : {},
        .format(values[2]).encode())
    usb_cdc.data.write('"humidity" : {}
        .format(values[3]).encode())
    usb_cdc.data.write('}
        .encode())

# init time tracking
last_data = last_beat = time.monotonic()

# loop forever!
while True:
    current_time = time.monotonic()

    # data
    if current_time - last_data > DATA_RATE:
        data = (scd.CO2, bme.pressure, bme.temperature, bme.humidity)
        usb_cdc.data.reset_output_buffer()
        if DATA_FORMAT == "CSV":
            send_csv_data(data)
```

©Adafruit Industries
elif DATA_FORMAT == "JSON":
    send_json_data(data)
else:
    usb_cdc.data.write(b"Unknown data format.\n")
last_data = current_time

# heart beat
if BEAT_RATE and current_time - last_beat > BEAT_RATE:
    if pixel[0][0]:
        pixel.fill(0)
    else:
        pixel.fill(BEAT_COLOR)
    last_beat = current_time

Receiving Data

With either the Arduino or the CircuitPython code running on the Trinkey QT2040, the data is being sent out via a USB CDC serial port. This data can be received by anything that can open and talk to the serial port. Here we provide an example using Python and pySerial.

Install pySerial

The pySerial module is used to open and read from the serial port in Python. If this module is not already installed on your setup, go here:

https://adafru.it/S6E

Which Serial Port?

The Arduino example sends out data using the typical `Serial.print()` command. This serial port should show up in the place you’d look for the Arduino Serial Monitor.

The CircuitPython example uses a secondary serial port and data is sent using `usb_cdc.data.write()`. This is different than the serial port where `print()` output shows up. There should be two serial ports that show up when running the CircuitPython example. The data is most likely on the second one.

The serial port location will likely be different between the Arduino and CircuitPython code.
Read CSV String

Here is an example of how to open the serial port and read the incoming CSV data:

```python
# SPDX-FileCopyrightText: 2021 Carter Nelson for Adafruit Industries
# SPDX-License-Identifier: MIT

import serial

# open serial port (NOTE: change location as needed)
ss = serial.Serial("/dev/ttyACM0")

# read string
_ = ss.readline() # first read may be incomplete, just toss it
raw_string = ss.readline().strip().decode()

# create list of floats
data = [float(x) for x in raw_string.split(',')]  # Be sure to change receiving code as needed to point to proper serial port location.

# print them
print("CO2 =", data[0])
print("pressure =", data[1])
print("temperature =", data[2])
print("humidity =", data[3])
```

Read JSON String

Here is an example of how to open the serial port and read the incoming JSON data:

```python
# SPDX-FileCopyrightText: 2021 Carter Nelson for Adafruit Industries
# SPDX-License-Identifier: MIT

import json
import serial

# open serial port (NOTE: change location as needed)
ss = serial.Serial("/dev/ttyACM0")

# read string
_ = ss.readline() # first read may be incomplete, just toss it
raw_string = ss.readline().strip().decode()

# load JSON
data = json.loads(raw_string)

# print data
print("CO2 =", data['CO2'])
print("pressure =", data['pressure'])
print("temperature =", data['temperature'])
print("humidity =", data['humidity'])
```
Direct Reading with U2IF

This approach allows for direct reading of the sensors. Instead of reading and parsing from a serial stream, the sensor libraries are used directly in Python running on the host PC. This is done by having [specialized firmware called U2IF](https://adafru.it/Sje) running on the Trinkey QT2040. On the host PC, [Blinka](https://adafru.it/EQa) is installed to provide access to the Trinkey running U2IF and allow use of CircuitPython libraries.

This approach has a lot of setup. However, if your goal is to get sensor values into a Python application running on the host PC, this approach offers the most direct route.

Install U2IF Firmware onto Trinkey

This step is pretty easy. Just download the firmware file and copy to Trinkey. To download a copy of the U2IF firmware for the Trinkey QT2040, go here:

[U2IF Firmware for Trinkey QT2040](https://adafru.it/XAM)

To install the UF2 file:

1. Put the Trinkey QT2040 in bootloader mode by holding the BOOT button while pressing the RST (reset) button.
3. Drag the UF2 file to the RPI-RP2 folder.
4. Once copied, board should reset and code is now running.

Install Blinka onto PC

This step has more to it. The installation process is different for each operating system. Use this guide to install Blinka and the associated supporting software on to your PC:

[Setup on PC](https://adafru.it/XAN)

Make sure the post install checks pass before proceeding. Skip the check for the Pico, since were using a different board:
Install Libraries onto PC

Now we can install the specific CircuitPython libraries needed for talking to the sensors. For the SCD40, install this library:

SCD4x Library Install
https://adafruit.it/XAO

For the BME280, install this library:

BME280 Library Install
https://adafruit.it/FUP

Read Sensors via Python

Once all the setup is done, actually reading the sensors is easy.

Make sure you've set the BLINKA_U2IF environment variable.

Here is a basic "hello world" example of simply reading and printing the sensor values in a loop.

```python
# SPDX-FileCopyrightText: 2021 Carter Nelson for Adafruit Industries
# SPDX-License-Identifier: MIT
import time
import board
import adafruit_scd4x
from adafruit_bme280 import basic as adafruit_bme280

scd = adafruit_scd4x.SCD4X(board.I2C())
scd.start_periodic_measurement()

bme = adafruit_bme280.Adafruit_BME280_I2C(board.I2C())

while True:
    time.sleep(5)
    print("CO2 =", scd.CO2)
    print("Pressure = {:.1f} hPa".format(bme.pressure))
    print("Temperature = {:.1f} degC".format(bme.temperature))
    print("Humidity = {:.1f}%".format(bme.humidity))
```
Make sure you've set the BLINKA_U2IF environment variable.

Running that code should produce output similar to this: