# Table of Contents

## Overview 3

### Adafruit IO Setup 3
- Creating the Digital Feed 4
- Adding the Gauge Block 5

### Arduino Wiring 6

### Arduino Setup 7

### Arduino Network Config 8
- WiFi Config 8
- FONA Config 9
- Ethernet Config 9

### Arduino Code 10

### Python Wiring 13
- Parts 13
- Wiring 14

### Python Setup 15
- Update your Pi and Python 15
- Make sure you’re using Python 3! 16
- Install Python Libraries 16
- Installing Adafruit Blinka Library 16
- Installing Adafruit IO Python Library 17
- Downloading Example Code 17

### Python Code 18
- Code 18
- Running the Code 19
- Code 19

### Adafruit IO FAQ 20
- Encountering an issue with your Adafruit IO Arduino Project? 20
Overview

This guide is part of a series of guides that cover the basics of using Adafruit IO. It will show you how to send momentary button press data to Adafruit IO.

If you haven't worked your way through the Adafruit IO feed and dashboard basics guides, you should do that before continuing with this guide so you have a basic understanding of Adafruit IO.

- Adafruit IO Basics: Feeds
- Adafruit IO Basics: Dashboards

You should go through the setup guides associated with your selected set of hardware, and make sure you have internet connectivity with the device before continuing. The following links will take you to the guides for your selected platform.

- Adafruit Feather HUZZAH ESP8266 Setup Guide

If you have went through all of the prerequisites for your selected hardware, you are now ready to move on to the Adafruit IO setup steps that are common between all of the hardware choices for this project. Let's get started!

Adafruit IO Setup

The first thing you will need to do is to login to Adafruit IO and visit the Settings page.

Click the VIEW AIO KEY button to retrieve your key.
A window will pop up with your Adafruit IO. Keep a copy of this in a safe place. We'll need it later.

Creating the Digital Feed

Next, you will need to create a feed called Digital. If you need help getting started with creating feeds on Adafruit IO, check out the Adafruit IO Feed Basics guide (https://adafruit.io).
Adding the Gauge Block

Next, add a new Gauge block to a new or existing dashboard. Name the block whatever you would like, and give it a max value of 1 and a min value of 0. Make sure you have selected the Digital feed as the data source for the gauge.

If you need help getting started with Dashboards on Adafruit IO, check out the Adafruit IO Dashboard Basics guide (https://adafruit.io/f5m).

When you are finished editing the form, click Create Block to add the new block to the dashboard.
Next, we will look at wiring the circuit.

Arduino Wiring

You will need the following parts for this tutorial:

- 1x Adafruit IO compatible Feather
- 3x jumper wires
- 1x 10k resistor
- 1x momentary button

You will need to connect the following pins to the button and 10k resistor:

- Feather GND to one side of the momentary button
- Feather Pin 5 to the other side of the momentary button
- Feather 3V to one leg of a 10k resistor
- The second leg of the 10k resistor to the same side of the momentary button as Pin 5

Note: Resistors are not polarized, so the 10k resistor can be connected to the circuit in either direction.
Arduino Setup

You should go through the setup guides associated with your selected set of hardware, and make sure you have internet connectivity with the device before continuing. The following links will take you to the guides for your selected platform.

• Adafruit Feather HUZZAH ESP8266 Setup Guide

You will need to make sure you have at least version 2.3.1 of the Adafruit IO Arduino library installed before continuing. You will also need the Arduino HTTP library and Adafruit MQTT library so check the setup guide above to get all set up!

For this example, you will need to open the adafruitio_06_digital_in example in the Adafruit IO Arduino library.
Next, we will look at the network configuration options in the sketch.

**Arduino Network Config**

To configure the network settings, click on the config.h tab in the sketch. You will need to set your Adafruit IO username in the IO_USERNAME define, and your Adafruit IO key in the IO_KEY define.

```
// visit io.adafruit.com if you need to create an account,
// or if you need your Adafruit IO key.
#define IO_USERNAME "uniontownlabs"
#define IO_KEY "..."
```

**WiFi Config**

WiFi is enabled by default in config.h so if you are using one of the supported WiFi boards, you will only need to modify the WIFI_SSID and WIFI_PASS options in the config.h tab.
FONA Config

If you wish to use the FONA 32u4 Feather to connect to Adafruit IO, you will need to first comment out the WiFi support in config.h

```c
#define WIFI_SSID     "Test WiFi"
#define WIFI_PASS     "my wifi password"

// comment out the following two lines if you are using fona or ethernet
#include "AdafruitIO_WiFi.h"
AdafruitIO_WiFi io(IO_USERNAME, IO_KEY, WIFI_SSID, WIFI_PASS);
```

Next, remove the comments from both of the FONA config lines in the FONA section of config.h to enable FONA support.

```c
#include "AdafruitIO_FONA.h"
AdafruitIO_FONA io(IO_USERNAME, IO_KEY);
```

Ethernet Config

If you wish to use the Ethernet Wing to connect to Adafruit IO, you will need to first comment out the WiFi support in config.h

```c
#include "AdafruitIO_WiFi.h"
#include "AdafruitIO_FONA.h"
```

©Adafruit Industries
Next, remove the comments from both of the Ethernet config lines in the Ethernet section of config.h to enable Ethernet Wing support.

Next, we will look at how the example sketch works.

Arduino Code

The adafruitio_06_digital_in example uses digital pin 5 by default on all boards, and that can be modified if needed by changing the BUTTON_PIN define.

Note: If you are using the WICED Feather, you will need to change the BUTTON_PIN define to PC5 instead of the default setting of 5.

The next chunk of code sets up two boolean variables to track button state, and an Adafruit IO Feed instance for a feed called digital.
// button state
bool current = false;
bool last = false;

// set up the 'digital' feed
AdafruitIO_Feed *digital = io.feed("digital");

In the setup function, we set the BUTTON_PIN as a digital input, and connect to Adafruit IO. The code will wait until you have a valid connection to Adafruit IO before continuing with the sketch. If you have any issues connecting, check config.h for any typos in your username or key.

```cpp
void setup() {
    // set button pin as an input
    pinMode(BUTTON_PIN, INPUT);

    // start the serial connection
    Serial.begin(115200);

    // wait for serial monitor to open
    while(! Serial);

    // connect to io.adafruit.com
    Serial.print("Connecting to Adafruit IO");
    io.connect();

    // wait for a connection
    while(io.status() < AIO_CONNECTED) {
        Serial.print(".");
        delay(500);
    }

    // we are connected
    Serial.println();
    Serial.println(io.statusText());
}
```

Next, we have the main loop() function. The first line of the loop function calls `io.run();` this line will need to be present at the top of your loop in every sketch. It helps keep your device connected to Adafruit IO, and processes any incoming data.

```cpp
void loop() {
    // io.run(); is required for all sketches.
    // it should always be present at the top of your loop
    // function. it keeps the client connected to
    // io.adafruit.com, and processes any incoming data.
    io.run();
}
```

The next chunk of code inside the loop() checks the current state of the button, and saves the state of the button in the current variable. Because we are using a pullup resistor, we will need to flip the button state.
If the button state is LOW it means the button is pressed, so we set `current = true;`. If the button state is HIGH it means the button is released, so we set `current = false;`.

We then check if the current button state is equal to the last button state. If it is equal, we will return early and not continue with the rest of the loop.

```c
// grab the current state of the button.
// we have to flip the logic because we are
// using a pullup resistor.
if(digitalRead(BUTTON_PIN) == LOW)
  current = true;
else
  current = false;

// return if the value hasn't changed
if(current == last)
  return;
```

The final chunk of the `loop()` function prints the current value to the Arduino Serial Monitor, and sends the current value to the digital feed on Adafruit IO. We also set `last = current;` so we can tell if the state of the button has changed in the next run of the loop.

```c
// save the current state to the 'digital' feed on adafruit io
Serial.print("sending button -> ");
Serial.println(current);
digital-&gt;save(current);

// store last button state
last = current;
```

Upload the sketch to your board, and open the Arduino Serial Monitor. Your board should now connect to Adafruit IO.

```
Connecting to Adafruit IO....
Adafruit IO connected.
```

You can now press the button, and you should see button presses being sent to Adafruit IO.

```
sending button -&gt; 1
sending button -&gt; 0
sending button -&gt; 1
sending button -&gt; 0
```
Check your dashboard on Adafruit IO, and you should see the gauge respond to button presses.

Python Wiring

Parts

Raspberry Pi 3 - Model B+ - 1.4GHz Cortex-A53 with 1GB RAM
The Raspberry Pi 3 Model B is the most popular Raspberry Pi computer made, and the Pi Foundation knows you can always make a good thing better! And what could make the Pi 3...

https://www.adafruit.com/product/3775

If you’re following along with a Raspberry Pi (https://adafruit.it/ejq), we're going to use a T-Cobbler Plus for the IO Basics Projects. This add-on prototyping board lets you easily connect a Raspberry Pi (Raspberry Pi Model Zero, A+, B+, Pi 2, Pi 3) to a solderless breadboard.
1 x Jumper Wires
Breadboarding wire bundle.

1 x Button
Tactile Switch Buttons (12mm square, 6mm tall) x 10 pack

Assembled Pi T-Cobbler Plus - GPIO Breakout
This is the assembled version of the Pi T-Cobbler Plus. It only works with the Raspberry Pi Model Zero, A+, B+, Pi 2, Pi 3 & Pi 4! (Any Pi with 2x20...
https://www.adafruit.com/product/2028

You'll need to make the following connections:

- Pi GND to a common ground rail.
- Pi GND to one side of the momentary button.
- Pi Pin 12 to the other side of the momentary button.

Next, proceed to the Python Setup Page (https://adafru.it/BMB) of this guide.
Python Setup

If you're following along with a Raspberry Pi, Beaglebone or any other supported small linux computer, we'll use a special library called adafruit_blinka (https://adafruit.it/BJS) (named after Blinka, the CircuitPython mascot (https://adafruit.it/BJT)) to provide the layer that translates the CircuitPython hardware API to whatever library the Linux board provides. It's CircuitPython, on Pi!

Update your Pi and Python

The latest Raspbian (currently this is `Stretch`) is required for the installation of Adafruit IO + Blinka.

In this page we'll assume you've already gotten your Raspberry Pi up and running and can log into the command line.

Go ahead and ssh into your Raspberry Pi via terminal or a ssh client:

```
ssh pi@raspberrypi.local
```

Run the standard updates:

```
sudo apt-get update
```
```
sudo apt-get upgrade
```

and

```
sudo pip3 install --upgrade setuptools
```

### Make sure you're using Python 3!

The default python on your computer may not be python 3. Python 2 is officially discontinued and all our libraries are Python 3 only.

We'll be using **python3** and **pip3** in our commands, use those versions of python and pip to make sure you're using 3 and not 2.

### Install Python Libraries

### Installing Adafruit Blinka Library

Now you're ready to install all the python support:

Run the following command to install the Raspberry PI GPIO library:

```
pip3 install RPI.GPIO
```

Run the following command to install adafruit_blinka

```
pip3 install adafruit-blinka
```
The computer will install a few different libraries such as adafruit-pureio (our ioctl-only i2c library), spidev (for SPI interfacing), Adafruit-GPIO (for detecting your board) and of course adafruit-blinka.

Installing Adafruit IO Python Library

We'll also need to install the Adafruit IO Python Client Library (https://adafruit.io) to communicate with Adafruit IO.

Run the following command to install the Adafruit IO Client for Python:

```bash
pip3 install adafruit-io
```

If the installation gives you 'insufficient permissions' errors, add 'sudo' before the call to pip3.

Downloading Example Code

The example code is contained within the Python IO Client's examples/basics subdirectory.

Navigate to the root directory of your Pi:

```bash
cd ~
```

Then, download the latest version of the adafruit/io-client-python repository by running:

```bash
git clone https://github.com/adafruit/io-client-python.git
```

Navigate to that folder's example folder for the examples:
cd io-client-python/examples/basics/

That's it! We're all set up.

Next, let's upload some code and learn how it works.

---

**Python Code**

**Code**

Before we run the script at the bottom of this page, we'll need to change `ADAFRUIT_IO_USERNAME` and `ADAFRUIT_IO_KEY` to the username and key for your Adafruit IO account.

- If you need the AIO Key, navigate to [your Adafruit IO Profile](https://adafruit.io/BmD)

```python
# Set to your Adafruit IO key.  # Remember, your key is a secret, # so make sure not to publish it when you publish this code!
ADAFRUIT_IO_KEY = 'YOUR_AIO_KEY'

# Set to your Adafruit IO username.  # (go to https://accounts.adafruit.com to find your username)
ADAFRUIT_IO_USERNAME = 'YOUR_AIO_USERNAME'
```

We're going to set up an instance of the feed we created earlier:

```
digital = aio.create_feed(Feed(name="digital"))
```

In the `while True` loop, we're going to check the value of the button and send it to Adafruit IO. A delay (time.sleep()) has been added to avoid timing out by sending too many requests to Adafruit IO.

```python
while True:
    if not button.value:
        button_current = 1
    else:
        button_current = 0

    print('Button -&gt; ', button_current)
    aio.send(digital.key, button_current)

    # avoid timeout from adafruit io
    time.sleep(1)
```
Running the Code

Make sure you're within the /io-client-python/examples/basics directory.

If you're not sure which directory you're in, you can check this by running `pwd` and you should see the following output from your terminal:

```
~/io-client-python/examples/basics
```

Let's run the script. In your terminal, run:

```
python3 digital-in.py
```

You can now press the button, and you should see button presses being sent to Adafruit IO:

```
Button -&gt; 1
Button -&gt; 0
Button -&gt; 1
Button -&gt; 0
```

Check your dashboard on Adafruit IO, and you should see the gauge respond to button presses:

```
Code

```"""
'digital_in.py'
==================================
Example of sending button values to an Adafruit IO feed.
Author(s): Brent Rubell, Todd Treece
```
# Import standard python modules
import time

# import Adafruit Blinka
import board
import digitalio

# import Adafruit IO REST client.
from Adafruit_IO import Client, Feed, RequestError

# Set to your Adafruit IO key.
# Remember, your key is a secret,
# so make sure not to publish it when you publish this code!
ADAFRUIT_IO_KEY = 'YOUR_AIO_KEY'

# Set to your Adafruit IO username.
# (go to https://accounts.adafruit.com to find your username)
ADAFRUIT_IO_USERNAME = 'YOUR_AIO_USERNAME'

# Create an instance of the REST client.
aio = Client(ADAFRUIT_IO_USERNAME, ADAFRUIT_IO_KEY)

try: # if we have a 'digital' feed
digital = aio.feeds('digital')
except RequestError: # create a digital feed
    feed = Feed(name="digital")
digital = aio.create_feed(feed)

# button set up
button = digitalio.DigitalInOut(board.D12)
button.direction = digitalio.Direction.INPUT
button.pull = digitalio.Pull.UP
button_current = 0

while True:
    if not button.value:
        button_current = 1
    else:
        button_current = 0

    print('Button -> ', button_current)
aio.send(digital.key, button_current)

    # avoid timeout from adafruit io
time.sleep(1)

---

Adafruit IO FAQ

Encountering an issue with your Adafruit IO Arduino Project?

If you’re having an issue compiling, connecting, or troubleshooting your project, check this page first.

Don't see your issue? [Post up on the Adafruit IO Forum with your issue](https://adafru.it/plC).
I encounter the following error when compiling my sketch:

```
fatal error: Adafruit_MQTT.h: No such file or directory, #include "Adafruit_MQTT.h"
```

The Adafruit IO Arduino library is dependent on our Adafruit IO MQTT Library.

To resolve this error, from the Arduino IDE, navigate to the Manage Libraries... option in the Sketch -> Include Library menu.

My Serial Monitor prints "..." endlessly after the "Connecting to Adafruit IO" message

Your board is not connecting to Adafruit IO, but why? Let's find out:

First, check in `config.h` that you have the correct `IO_USERNAME`, `IO_KEY`, `WIFI_SSID`, and `WIFI_PASS` are set correctly.

Next, we're going to modify the while loop which waits for an IO connection in your sketch. Change the line in the status check loop from `Serial.println(.;` to `Serial.println(io.statusText());`
// wait for a connection
while(io.status() < AIO_CONNECTED) {
  Serial.println(io.statusText());
  delay(500);
}

Verify and re-upload the sketch. If you're receiving a Network disconnected error message, the board is not able to talk to the internet. Re-check your hardware, connections, and router settings.

If it's still not showing Adafruit IO connected, check the IO status on the Adafruit Status page (https://adafruit.it/Oc0) to make sure the service is online.

My data isn't displaying, is Adafruit IO's [service/MQTT/API] down?
   Possibly - you can check IO status on the Adafruit Status page (https://adafruit.it/Oc0).

Is my data being sent properly? Am I sending too much data?
   There's a monitor page built-into Adafruit IO (https://adafruit.it/DOK) which provides a live view of incoming data and error messages. Keep this page open while you send data to your Adafruit IO devices to monitor data and errors.