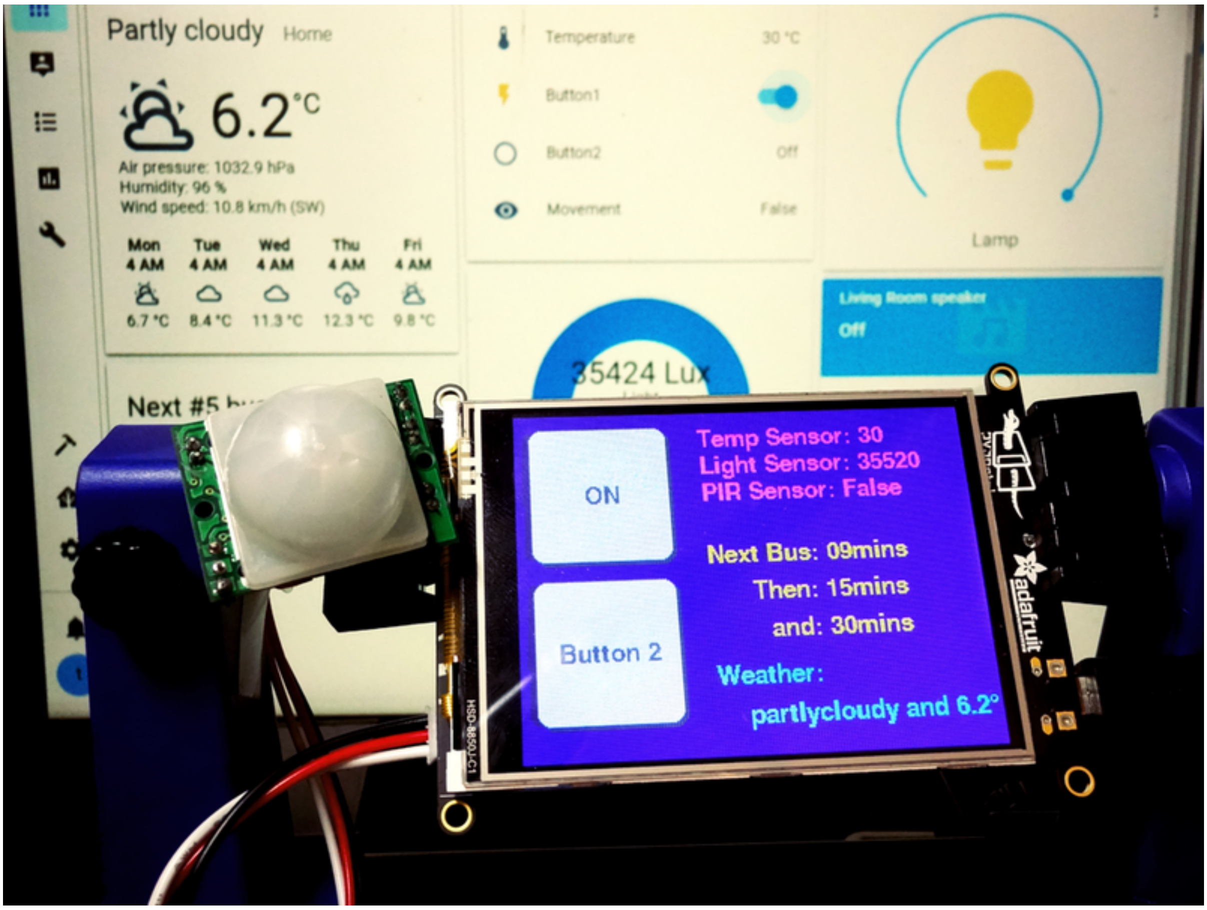
PyPortal MQTT Sensor Node/Control Pad for Home Assistant

Created by Richard Albritton



https://learn.adafruit.com/pyportal-mqtt-sensor-node-control-pad-home-assistant

Last updated on 2025-04-14 11:56:26 AM EDT

Table of Contents

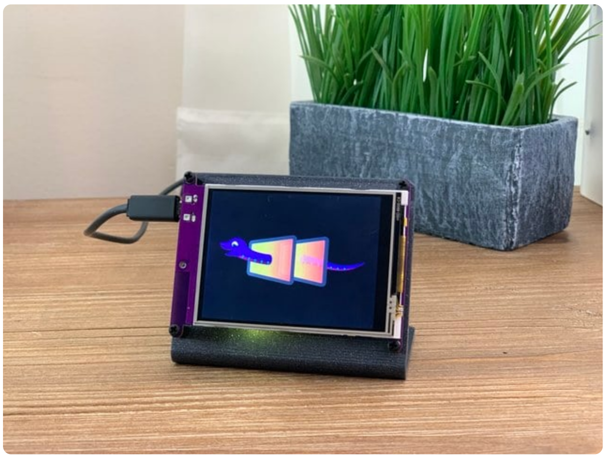
Overview

Using a computer or an app on a tablet is a good way to control all the things in your Smart Home, but sometimes it is overkill. So what if we had a small touchscreen device that would let you control just a few things that you need quick access to? Something like a super smart, but not too smart, light switch that uses much less power than a smartphone or tablet. Even better if it also had some sensors on it and could display some simple data on the screen like the weather forecast.

The PyPortal is perfect for this kind of work with its built-in 3.2″ TFT Touchscreen, ESP32 WiFi controller, plenty of storage, and a SAMD51 M4 processor chip at its core. So we will be using the PyPortal to build out a user interface that will talk to our Smart Home system using MQTT.

When finished, the PyPortal will read the temperature, light level, detect motion, and use virtual buttons to send data via MQTT to Home Assistant, so it can be used in Automations. Along with the sensors, we will be able to display data from Home Assistant on the PyPortal screen for quick updates.

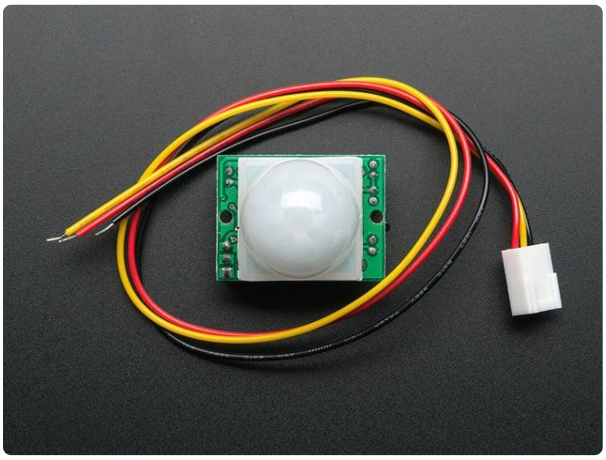
Parts



Adafruit PyPortal - CircuitPython Powered Internet Display

PyPortal, our easy-to-use IoT device that allows you to create all the things for the “Internet of Things” in minutes. Make custom touch screen interface...

https://www.adafruit.com/product/4116



PIR (motion) sensor

PIR sensors are used to detect motion from pets/humanoids from about 20 feet away (possibly works on zombies, not guaranteed). This one has an adjustable delay before firing (approx...

https://www.adafruit.com/product/189

STEMMA JST PH 2mm 3-Pin to Female Socket Cable - 200mm

This cable will let you turn a JST PH 3-pin cable port into 3 individual wires with high-quality 0.1" female header sockets on the end. We're carrying these to match up with...

https://www.adafruit.com/product/3894

USB cable - USB A to Micro-B

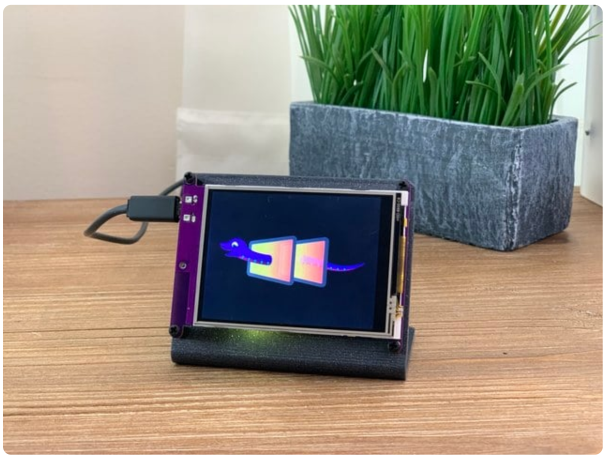
This here is your standard A to micro-B USB cable, for USB 1.1 or 2.0. Perfect for connecting a PC to your Metro, Feather, Raspberry Pi or other dev-board or...

https://www.adafruit.com/product/592

Things you will need

There are two parts to this project that include setting up the PyPortal along with configuring Home Assistant. Here are some things that you will want to be sure you have ready before you begin.

For the PyPortal



Adafruit PyPortal - CircuitPython Powered Internet Display

PyPortal, our easy-to-use IoT device that allows you to create all the things for the “Internet of Things” in minutes. Make custom touch screen interface...

https://www.adafruit.com/product/4116

1 x PIR (motion) sensor https://www.adafruit.com/product/189

PIR sensors are used to detect motion from pets/humanoids from about 20 feet away

1 x STEMMA JST PH 3-Pin to Female Socket Cable - 200mm https://www.adafruit.com/product/3894

This cable will let you connect the PIR motion sensor to one of the JST PH 3-pin connectors on the PyPortal.

1 x USB cable - USB A to Micro-B - 3 foot long https://www.adafruit.com/product/592

Use this USB cable to program and power your PyPortal.

For Home Assistant

You will need a Home Assistant server set up for this project to interface with. There are many ways to do this, but most people set up Home Assistant on a Raspberry Pi. If you need to set up your own Home Assistant server, have a look at the following guide that will help you set up Home Assistant along with an MQTT broker.

Set up Home Assistant with a Raspberry Pi

https://adafru.it/Ibd

If you already have Home Assistant set up and running, you will need to have the Mosquitto broker installed. You will also need to know the **Host URL**, **Username**, and **Password** for your **MQTT broker**.

We will also be editing the **configuration.yaml** file to add support for the PyPortal MQTT topics. So be sure that you can access and edit the **configuration.yaml** file.

Your WiFi network SSID and password will also be needed for the PyPortal to connect to the same local network as your Home Assistant server.

Connecting the Sensors

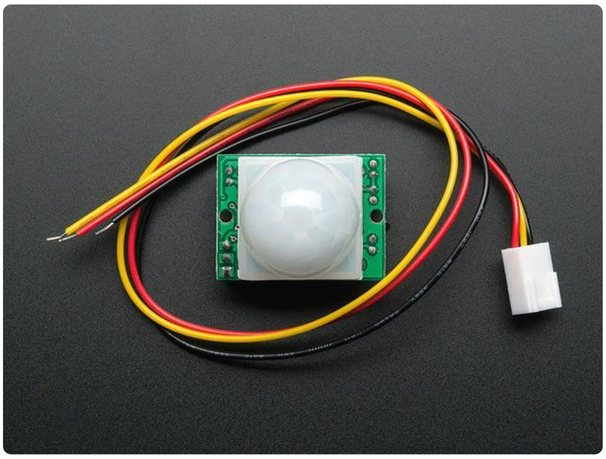
The PyPortal has a built-in Temperature and Light sensor that we will make good use of, but we can always add more. Motion sensors are a good way to tell if a person is in a particular room and so we will be using a passive infrared (or PIR) sensor as a "people detector".

On the PyPortal, there are two STEMMA 3-Pin JST connectors that make it easy to connect new sensors. Here is what you will need:

STEMMA JST PH 2mm 3-Pin to Female Socket Cable - 200mm

This cable will let you turn a JST PH 3-pin cable port into 3 individual wires with high-quality 0.1" female header sockets on the end. We're carrying these to match up with...

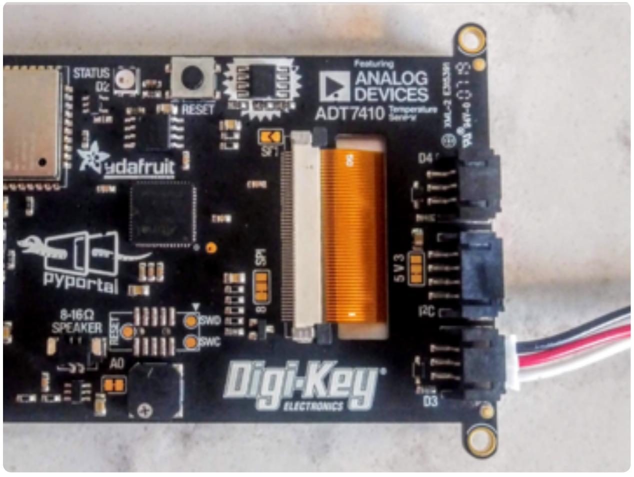
https://www.adafruit.com/product/3894

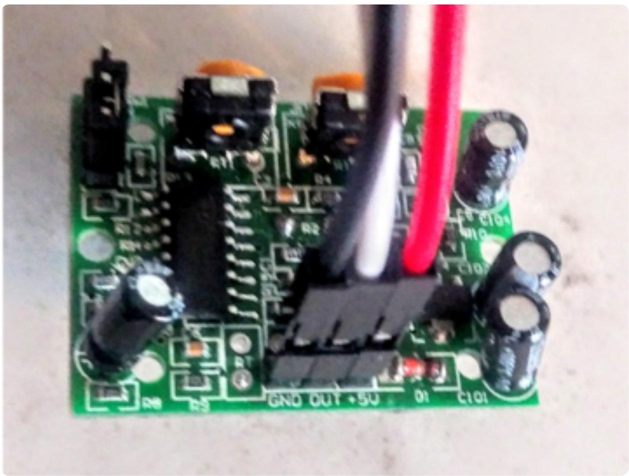


PIR (motion) sensor

PIR sensors are used to detect motion from pets/humanoids from about 20 feet away (possibly works on zombies, not guaranteed). This one has an adjustable delay before firing (approx...

https://www.adafruit.com/product/189





You will want to connect the STEMMA 3-Pin JST to the D3 port on the PyPortal.

The other end of the wires get connected to the PIR sensor.

Black wire to **GND**
White wire to **OUT**
Red wire to **+5V**

The Code

We will be using CircuitPython on the PyPortal. If you are not familiar with how to use CircuitPython or run into trouble, you may want to have a look at the following guide:

Adafruit PyPortal - IoT for CircuitPython: Install Circuitpython

https://adafru.it/EnM

Next, we will go over all of the code that will be running on the PyPortal.

CircuitPython Code

Setup PyPortal with CircuitPython

We'll need to get our PyPortal board setup so it can run CircuitPython code. Let's walk through these steps to get the latest version of CircuitPython onto your board

The Mu Python Editor

Mu is a simple Python editor that works with Adafruit CircuitPython hardware. It's written in Python and works on Windows, MacOS, Linux and Raspberry Pi. The serial console is built right in, so you get immediate feedback from your board's serial output! While you can use any text editor with your code, Mu makes it super simple.

Installing and Using the Mu Editor

https://adafru.it/ANO

Installing or upgrading CircuitPython

You should ensure you have CircuitPython 4.0 or greater on your board. Plug your board in with a known good data + power cable (not the cheesy USB cable that comes with USB power packs, they are power only). You should see a new flash drive pop up.

If the drive is **CIRCUITPY**, then open the **boot_out.txt** file to ensure the version number is 4.0 or greater.

You can download everything that you need for the PyPortal code by downloading the following Zip file and copying its contents to your PyPortals **CIRCUITPY** folder.

```
Adafruit CircuitPython 5.0.0-beta.0 on 2019-11-19; Adafruit PyPortal with samd51g19
```

Upload the Code and Files

Click on the **Download Project Bundle** button below to grab the main code and other files noted (except the library files) directly from GitHub (the repository is here (https://adafru.it/19EQ)). Drop the files onto the **CIRCUITPY** main (root) directory (with the font file in a **fonts** directory). The code will run properly when all of the files have been uploaded including libraries.

Use any text editor or favorite IDE to modify the code. We suggest using Mu as noted above.

```
# SPDX-FileCopyrightText: 2020 Anne Barela for Adafruit Industries
#
# SPDX-License-Identifier: MIT

import board
import displayio
import busio
from digitalio import DigitalInOut
from analogio import AnalogIn
import neopixel
import adafruit_adt7410
import adafruit_connection_manager
from adafruit_esp32spi import adafruit_esp32spi
from adafruit_esp32spi import adafruit_esp32spi_wifimanager
from adafruit_bitmap_font import bitmap_font
from adafruit_display_text.label import Label
from adafruit_button import Button
import adafruit_touchscreen
import adafruit_minimqtt.adafruit_minimqtt as MQTT

# ------------- WiFi ------------- #

# Get wifi details and more from a secrets.py file
try:
    from secrets import secrets
except ImportError:
    print("WiFi secrets are kept in secrets.py, please add them there!")
    raise

# If you are using a board with pre-defined ESP32 Pins:
esp32_cs = DigitalInOut(board.ESP_CS)
esp32_ready = DigitalInOut(board.ESP_BUSY)
esp32_reset = DigitalInOut(board.ESP_RESET)

spi = busio.SPI(board.SCK, board.MOSI, board.MISO)
esp = adafruit_esp32spi.ESP_SPIcontrol(spi, esp32_cs, esp32_ready, esp32_reset)
status_light = neopixel.NeoPixel(board.NEOPIXEL, 1, brightness=0.2)
wifi = adafruit_esp32spi_wifimanager.ESPSPI_WiFiManager(esp, secrets, status_light)

# ------- Sensor Setup ------- #
# init. the temperature sensor
i2c_bus = busio.I2C(board.SCL, board.SDA)
adt = adafruit_adt7410.ADT7410(i2c_bus, address=0x48)
adt.high_resolution = True
temperature = "blaa"
# init. the light sensor
light_sensor = AnalogIn(board.LIGHT)

# init. the motion sensor
movement_sensor = DigitalInOut(board.D3)

button1_state = 0
button2_state = 0

# ------------- Screen eliments ------------- #

display = board.DISPLAY

# Backlight function
def set_backlight(val):
    """Adjust the TFT backlight.
    :param val: The backlight brightness. Use a value between ``0`` and ``1``, where
``0`` is
                off, and ``1`` is 100% brightness.
    """
    val = max(0, min(1.0, val))
    try:
        board.DISPLAY.auto_brightness = False
    except AttributeError:
        pass
    board.DISPLAY.brightness = val


# Touchscreen setup
ts = adafruit_touchscreen.Touchscreen(
    board.TOUCH_XL,
    board.TOUCH_XR,
    board.TOUCH_YD,
    board.TOUCH_YU,
    calibration=((5200, 59000), (5800, 57000)),
    size=(320, 240),
)

# ---------- Set the font and preload letters ----------
# Be sure to put your font into a folder named "fonts".
font = bitmap_font.load_font("/fonts/Helvetica-Bold-16.bdf")
# This will preload the text images.
font.load_glyphs(b"abcdefghjiklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890-
()")

# ------------- User Inretface Eliments ------------- #

# Make the display context
splash = displayio.Group()
board.DISPLAY.root_group = splash

# Make a background color fill
color_bitmap = displayio.Bitmap(320, 240, 1)
color_palette = displayio.Palette(1)
color_palette[0] = 0x3D0068
bg_sprite = displayio.TileGrid(color_bitmap, x=0, y=0, pixel_shader=color_palette)
splash.append(bg_sprite)

buttons = []
# Default button styling:
BUTTON_WIDTH = 100
BUTTON_HEIGHT = 100
BUTTON_MARGIN = 10

# Button Objects
button_1 = Button(
    x=BUTTON_MARGIN,
    y=BUTTON_MARGIN,
    width=BUTTON_WIDTH,
    height=BUTTON_HEIGHT,
    label="Button 1",
    label_font=font,
    style=Button.SHADOWROUNDRECT,
    label_color=0x505050,
    fill_color=0x9E9E9E,
    outline_color=0x464646,
)
buttons.append(button_1)

button_2 = Button(
    x=BUTTON_MARGIN,
    y=BUTTON_MARGIN * 2 + BUTTON_HEIGHT,
    width=BUTTON_WIDTH,
    height=BUTTON_HEIGHT,
    label="Button 2",
    label_font=font,
    style=Button.SHADOWROUNDRECT,
    label_color=0x505050,
    fill_color=0x9E9E9E,
    outline_color=0x464646,
)
buttons.append(button_2)
```

```
for b in buttons:
    splash.append(b.group)

# Text Label Objects
temperature_label = Label(font, text="temperature", color=0xE300D2)
temperature_label.x = 130
temperature_label.y = 20
splash.append(temperature_label)

light_label = Label(font, text="lux", color=0xE300D2)
light_label.x = 130
light_label.y = 40
splash.append(light_label)

motion_label = Label(font, text="motion", color=0xE300D2)
motion_label.x = 130
motion_label.y = 60
splash.append(motion_label)

feed1_label = Label(font, text="MQTT feed1", color=0xE39300)
feed1_label.x = 130
feed1_label.y = 130
splash.append(feed1_label)

feed2_label = Label(font, text="MQTT feed2", color=0x00DCE3)
feed2_label.x = 130
feed2_label.y = 200
splash.append(feed2_label)

# ------------- MQTT Topic Setup ------------- #

mqtt_topic = "test/topic"
mqtt_temperature = "pyportal/temperature"
mqtt_lux = "pyportal/lux"
mqtt_PIR = "pyportal/pir"
mqtt_button1 = "pyportal/button1"
mqtt_button2 = "pyportal/button2"
mqtt_feed1 = "pyportal/feed1"
mqtt_feed2 = "pyportal/feed2"

# ------------- MQTT Functions ------------- #

# Define callback methods which are called when events occur
# pylint: disable=unused-argument, redefined-outer-name
def connect(client, userdata, flags, rc):
    # This function will be called when the client is connected
    # successfully to the broker.
    print("Connected to MQTT Broker!")
    print("Flags: {0}\n RC: {1}".format(flags, rc))


def disconnected(client, userdata, rc):
    # This method is called when the client is disconnected
    print("Disconnected from MQTT Broker!")


def subscribe(client, userdata, topic, granted_qos):
    # This method is called when the client subscribes to a new feed.
    print("Subscribed to {0} with QOS level {1}".format(topic, granted_qos))


def publish(client, userdata, topic, pid):
    # This method is called when the client publishes data to a feed.
    print("Published to {0} with PID {1}".format(topic, pid))


def message(client, topic, message):
    """Method callled when a client's subscribed feed has a new
    value.
    :param str topic: The topic of the feed with a new value.
    :param str message: The new value
    """
    print("New message on topic {0}: {1}".format(topic, message))
    if topic == "pyportal/feed1":
        feed1_label.text = "Next Bus: {}".format(message)
    if topic == "pyportal/feed2":
        feed2_label.text = "Weather: \n    {}".format(message)
    if topic == "pyportal/button1":
        if message == "1":
            buttons[0].label = "ON"
            buttons[0].selected = False
            print("Button 1 ON")
        else:
            buttons[0].label = "OFF"
            buttons[0].selected = True
            print("Button 1 OFF")


# ------------- Network Connection ------------- #

# Connect to WiFi
print("Connecting to WiFi...")
wifi.connect()
print("Connected to WiFi!")

pool = adafruit_connection_manager.get_radio_socketpool(esp)
ssl_context = adafruit_connection_manager.get_radio_ssl_context(esp)

# Set up a MiniMQTT Client
client = MQTT.MQTT(
    broker=secrets["broker"],
    port=1883,
    username=secrets["user"],
    password=secrets["pass"],
    socket_pool=pool,
    ssl_context=ssl_context,
)

# Connect callback handlers to client
client.on_connect = connect
client.on_disconnect = disconnected
client.on_subscribe = subscribe
client.on_publish = publish
client.on_message = message

print("Attempting to connect to %s" % client.broker)
client.connect()

print(
    "Subscribing to %s, %s, %s, and %s"
    % (mqtt_feed1, mqtt_feed2, mqtt_button1, mqtt_button2)
)
client.subscribe(mqtt_feed1)
client.subscribe(mqtt_feed2)
client.subscribe(mqtt_button1)
client.subscribe(mqtt_button2)

# ------------- Code Loop ------------- #
while True:
    # Poll the message queue
    client.loop()

    # Read sensor data and format
    light_value = lux = light_sensor.value
    light_label.text = "Light Sensor: {}".format(light_value)
    temperature = round(adt.temperature)
    temperature_label.text = "Temp Sensor: {}".format(temperature)
    movement_value = movement_sensor.value
    motion_label.text = "PIR Sensor: {}".format(movement_value)

    # Read display button press
    touch = ts.touch_point
    if touch:
        for i, b in enumerate(buttons):
            if b.contains(touch):
                print("Sending button%d pressed" % i)
                if i == 0:
                    # Toggle switch button type
                    if button1_state == 0:
                        button1_state = 1
                        b.label = "ON"
                        b.selected = False
                        print("Button 1 ON")
                    else:
                        button1_state = 0
                        b.label = "OFF"
                        b.selected = True
                        print("Button 1 OFF")
                    print("Sending button 1 state: ")
                    client.publish(mqtt_button1, button1_state)
                    # for debounce
                    while ts.touch_point:
                        print("Button 1 Pressed")
                if i == 1:
                    # Momentary button type
                    b.selected = True
                    print("Sending button 2 state: ")
                    client.publish(mqtt_button2, 1)
                    # for debounce
                    while ts.touch_point:
                        print("Button 2 Pressed")
                    print("Button 2 reliced")
                    print("Sending button 2 state: ")
                    client.publish(mqtt_button2, 0)
                    b.selected = False

    # Publish sensor data to MQTT
    print("Sending light sensor value: %d" % light_value)
    client.publish(mqtt_lux, light_value)

    print("Sending temperature value: %d" % temperature)
    client.publish(mqtt_temperature, temperature)

    print("Sending motion sensor value: %d" % movement_value)
    client.publish(mqtt_PIR, "{}".format(movement_value))
```

The Font File

You will also need to create a folder named **fonts** on the PyPortal **CIRCUITPY** drive with the following bitmap font in it, available for download in the project zip or the GitHub repo (https://adafru.it/Ibe):

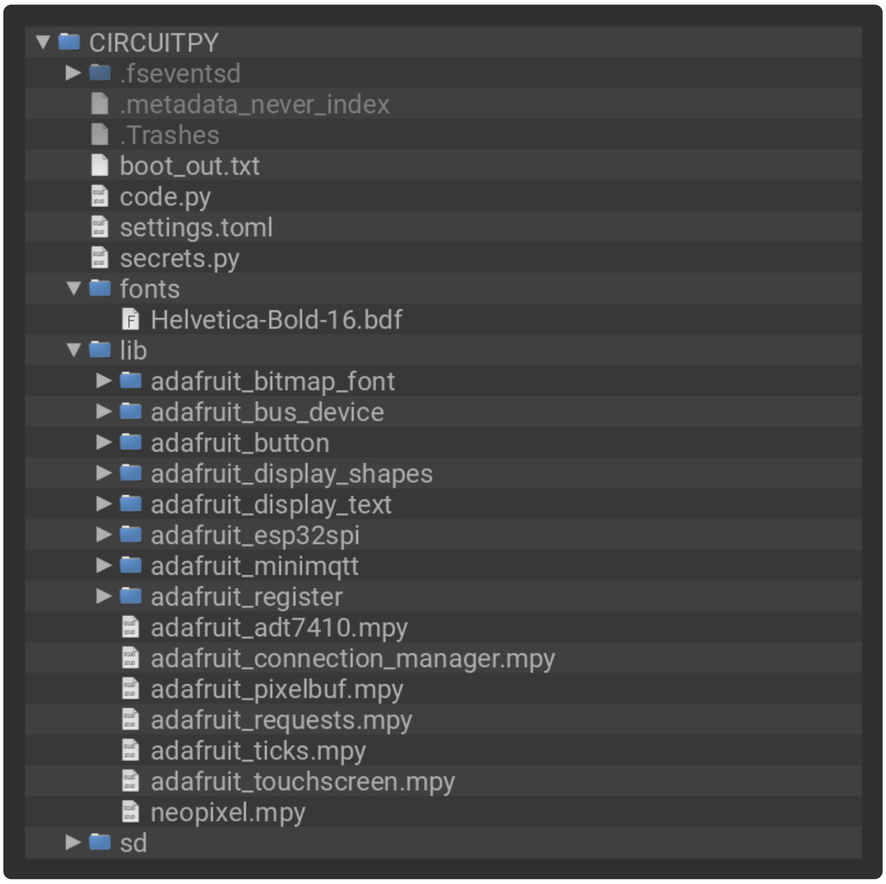
- **Helvetica-Bold-16.bdf**

Settings.toml file

You will also need a file named **settings.toml** that will contain your WiFi credentials as well as your MQTT server credentials.

```
CIRCUITPY_WIFI_SSID="your-wifi-ssid"
CIRCUITPY_WIFI_PASSWORD="your-wifi-password"
mqtt_broker="your-mqtt-broker-url-or-ip"
mqtt_username="your-mqtt-broker-username"
mqtt_password="your-mqtt-broker-password"
```

When you are finished the **CIRCUITPY** drive should look something like this:



```
▼ CIRCUITPY
    ▶ .fseventsd
      .metadata_never_index
      .Trashes
      boot_out.txt
      code.py
      settings.toml
      secrets.py
    ▼ fonts
        Helvetica-Bold-16.bdf
    ▼ lib
        ▶ adafruit_bitmap_font
        ▶ adafruit_bus_device
        ▶ adafruit_button
        ▶ adafruit_display_shapes
        ▶ adafruit_display_text
        ▶ adafruit_esp32spi
        ▶ adafruit_minimqtt
        ▶ adafruit_register
          adafruit_adt7410.mpy
          adafruit_connection_manager.mpy
          adafruit_pixelbuf.mpy
          adafruit_requests.mpy
          adafruit_ticks.mpy
          adafruit_touchscreen.mpy
          neopixel.mpy
    ▶ sd
```

Required Libraries

You will need the following Libraries installed for this project to run:

- adafruit_adt7410.mpy
- adafruit_connection_manager.mpy
- adafruit_pixelbuf.mpy
- adafruit_ticks.mpy
- adafruit_logging.mpy
- adafruit_minimqtt.mpy
- adafruit_pyportal.mpy
- adafruit_requests.mpy

- adafruit_touchscreen.mpy
- neopixel.mpy
- **adafruit_bitmap_font**
- **adafruit_bus_device**
- **adafruit_display_shapes**
- **adafruit_button**
- **adafruit_display_text**
- **adafruit_esp32spi**
- **adafruit_minimqtt**
- **adafruit_io**
- **adafruit_register**

Code Breakdown

Now we will have a look at the critical parts of the CircuitPython code so you know how it works and how you can change things to make it work the way you want it to. We are going to skip over a few sections that handle things like WiFi connection via the on-board ESP32 and other things that need to be included but are best not changed.

There are plenty of places for you to customize the layout of this display and that is what this section is all about.

Sensor Setup

The PyPortal has a temperature and light sensor already attached to it as well as two IO connectors. One of these connectors will have a PIR Sensor attached to detect motion.

Here a few global variables are set up that represent sensor readings and button states for later use. For input data, we will be getting information from the ADT7410 Temperature sensor, the onboard light sensor, PIR motion sensor, and two touchscreen buttons.

```
# ------- Sensor Setup ------- #
# init. the temperature sensor
i2c_bus = busio.I2C(board.SCL, board.SDA)
adt = adafruit_adt7410.ADT7410(i2c_bus, address=0x48)
adt.high_resolution = True
temperature = ""
# init. the light sensor
light_sensor = AnalogIn(board.LIGHT)

# init. the motion sensor
movement_sensor = DigitalInOut(board.D3)
```

```
button1_state = 0
button2_state = 0
```

You can add more inputs using the D4 connector or connecting to the four pin i2c bus. Then simply add the setup variables to this section.

Bitmap Fonts

This section will let you load a bitmap font and pre-load the letter and number glyphs to speed up information updates.

The font can be changed out with any other font but you will need to update the file path for your font in the following line of code:

`font = bitmap_font.load_font("/fonts/Helvetica-Bold-16.bdf")`

For this example I have created a folder named **fonts** that contains the **Helvetica-Bold-16.bdf** file.

```
# ---------- Set the font and preload letters ----------
# Be sure to put your font into a folder named "fonts".
font = bitmap_font.load_font("/fonts/Helvetica-Bold-16.bdf")
# This will preload the text images.
font.load_glyphs(b'abcdefghjiklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890-
()')
```

User interface Elements

Since we have many types of information to display, we will make use of Groups and Sub Groups to keep track of all our UI Elements. This will basically let us set up a single group with elements that can be displayed all at once.

The group that was set up for this example is called `splash` and it is rendered on the screen when the following code is called:

`board.DISPLAY.root_group = splash`

Elements can be added to this group with the following code where **yourElement** is replaced with the name of the element you are adding.

`splash.append(yourElement)`

Click here for more information about Display Groups and Subgroups.

https://adafru.it/Ibf

You can also change the background color by changing the HEX color code assigned to **color_palette[0]**.

```
# ------------- User Inretface Eliments ------------- #

# Make the display context
splash = displayio.Group()
board.DISPLAY.root_group = splash

# Make a background color fill
color_bitmap = displayio.Bitmap(320, 240, 1)
color_palette = displayio.Palette(1)
color_palette[0] = 0x3D0068
bg_sprite = displayio.TileGrid(color_bitmap, x=0, y=0,
                               pixel_shader=color_palette)
splash.append(bg_sprite)
```

Display Buttons

This example sets up two buttons that will react when touched. These buttons will be used to send signals to Home Assistant via MQTT. Each button is added to a Subgroup named `buttons`. This is then added to the `splash` Group.

The Button element accepts parameters for the `x` and `y` position along with the `height`, `width`, `label`, `label_font`, `label_color`, `fill_color`, `outline_color`, and `style`.

First we set some style variables for the `BUTTON_WIDTH`, `BUTTON_HEIGHT`, and `BUTTON_MARGIN`.

Then we define the button object using `Button()` and filling in the appropriate parameters.

Once the buttons are all set up with unique names, they are added to the `buttons` group and that group is added to the `splash` group.

```
buttons = []
# Default button styling:
BUTTON_WIDTH = 100
BUTTON_HEIGHT = 100
BUTTON_MARGIN = 10

# Button Objects
button_1 = Button(x=BUTTON_MARGIN, y=BUTTON_MARGIN,
                  width=BUTTON_WIDTH, height=BUTTON_HEIGHT,
                  label="Button 1", label_font=font, style=Button.SHADOWROUNDRECT,
label_color=0x505050,
                  fill_color=0x9e9e9e, outline_color=0x464646)
buttons.append(button_1)

button_2 = Button(x=BUTTON_MARGIN, y=BUTTON_MARGIN*2+BUTTON_HEIGHT,
                  width=BUTTON_WIDTH, height=BUTTON_HEIGHT,
                  label="Button 2", label_font=font, style=Button.SHADOWROUNDRECT,
label_color=0x505050,
                  fill_color=0x9e9e9e, outline_color=0x464646)
buttons.append(button_2)
```

```
for b in buttons:
    splash.append(b.group)
```

You can add more buttons by defining a new `Button()` object with a new name and adding it to the `buttons` group like this:

```
button_3 = Button(x=BUTTON_MARGIN, y=BUTTON_MARGIN*3+BUTTON_HEIGHT*2,
width=BUTTON_WIDTH, height=BUTTON_HEIGHT, label="Button 3",
label_font=font, style=Button.SHADOWROUNDRECT, label_color=0x505050,
fill_color=0x9e9e9e, outline_color=0x464646)
```

```
buttons.append(button_3<span)
```

Though you may want to change the `BUTTON_HEIGHT` so that this third button will fit under the other two buttons.

Click here for more information on the Button( )

https://adafru.it/FiB

Label text

Now we want to set up the text areas that will be used to display sensor data along with feed data from Home Assistant.

Just like the buttons, we need to create a unique Label object using the `Label( )` function.

```
light_label = Label(font, text="lux", color=0xE300D2)
```

This function needs to know the following:

- **font** - what font are you using for this label?
- **text** - the text that you want to populate this label with
- **color** - what color do you want this text to be?

We will also set the `x` and `y` position of this label after it is created.

```
light_label.x = 130
```

```
light_label.y = 40
```

Next we add the new label to the `splash` group.

```
splash.append(light_label)
```

```
# Text Label Objects
temperature_label = Label(font, text="temperature", color=0xE300D2)
temperature_label.x = 130
temperature_label.y = 20
splash.append(temperature_label)

light_label = Label(font, text="lux", color=0xE300D2)
light_label.x = 130
light_label.y = 40
splash.append(light_label)

motion_label = Label(font, text="motion", color=0xE300D2)
motion_label.x = 130
motion_label.y = 60
splash.append(motion_label)

feed1_label = Label(font, text="MQTT feed1", color=0xE39300)
feed1_label.x = 130
feed1_label.y = 130
splash.append(feed1_label)

feed2_label = Label(font, text="MQTT feed2", color=0x00DCE3)
feed2_label.x = 130
feed2_label.y = 200
splash.append(feed2_label)
```

Again, you can add more Label objects if you like. Just change the **x** and **y** numbers to ensure that everything fits nicely.

https://adafru.it/KDW

Setting the MQTT Topics

This is a list of all feeds that your PyPortal will interact with. To keep things organized we are starting with the group feed `pyportal`.

```
# ------------- MQTT Topic Setup ------------- #

mqtt_topic = 'test/topic'
mqtt_temperature = 'pyportal/temperature'
mqtt_lux = 'pyportal/lux'
mqtt_PIR = 'pyportal/pir'
mqtt_button1 = 'pyportal/button1'
mqtt_button2 = 'pyportal/button2'
mqtt_feed1 = 'pyportal/feed1'
mqtt_feed2 = 'pyportal/feed2'
```

You can always add more feeds if you want.

The nice thing about MQTT is that a client device like this one can create a new feed simply by requesting one.

MQTT Functions

This is where we set up to do all of the connection and interactions with the MQTT server. Most of these functions should be kept the same unless you want to do a specific thing when say your PyPortal has connected to the MQTT server.

The important part of this section is the `message` function that handles incoming MQTT data from feeds that you are subscribed to.

When new data is posted to a feed, that data is sent to the PyPortal if it is a feed that this device has subscribed to. The `message` function captures the feed `topic` and `message` so that it can be passed to the code loop or otherwise acted on within the function.

This example looks to see if the message is from one of the topics that we want to display.

Next if the topic is `pyportal/feed1` or `pyportal/feed2`, it will format the message and set the text for the appropriate Label to the data from the revived message.

If the topic is for `pyportal/button1`, the message is filtered and used to set the `button1` , otherwise known as `buttons[0]`, state to match the new data. This is used to show how you would link switch type objects so that they all represents the current state of that MQTT feed.

In other words, the switch on the PyPortal will always be in the same state as the switch in Home Assistant so long as they both get data from the same feed.

> It is not recommended to use a large volume of code in the message function, as it is run frequently. Try to add the minimum code needed to process your incoming message, and try not to run code here that is not related to a subscribed feed.

```
def message(client, topic, message):
    """Method callled when a client's subscribed feed has a new
    value.
    :param str topic: The topic of the feed with a new value.
    :param str message: The new value
    """
    print('New message on topic {0}: {1}'.format(topic, message))
    if topic == "pyportal/feed1":
        feed1_label.text = 'Next Bus: {}'.format(message)
    if topic == "pyportal/feed2":
        feed2_label.text = 'Weather: \n    {}'.format(message)
    if topic == "pyportal/button1":
        if message == "1":
            buttons[0].label="ON"
            buttons[0].selected = False
            print("Button 1 ON")
        else:
            buttons[0].label="OFF"
            buttons[0].selected = True
            print("Button 1 OFF")
```

If you are subscribing to more feeds, you will want to add code here to process that message with the following code where `MyMessageTopic` represents the feed topic that you have subscribed to and `newFeed_label` represents a `label` object.

```
if topic == "MyMessageTopic":
  newFeed_label.text = 'New Feed: \n {}'.format(message)
```

For more information on MQTT functions and message handling, click here.

https://adafru.it/Ibg

Subscribing to the feeds

Now we skip over the network connection handling and get to where we actually tell our MQTT server what topics we would like to subscribe to.

Basically, if you want the PyPortal to be updated with any information from the MQTT server, you will need to subscribe to that topic.

```
print('Subscribing to %s, %s, and %s' % (mqtt_feed1, mqtt_feed2, mqtt_button1))
client.subscribe(mqtt_feed1)
client.subscribe(mqtt_feed2)
client.subscribe(mqtt_button1)
```

The Loop

Now we are into the code loop and the first thing we want to run is `client.loop()` witch checks for new MQTT message updates.

Next we are going to read some sensors, assign their values to a variable, and update the relevant label text by running:

`light_label.text = 'Light Sensor: {}'.format(light_value)`

for each of the inputs except for the display buttons.

```
# ------------- Code Loop ------------- #
while True:
    # Poll the message queue
    client.loop()

    # Read sensor data and format
    light_value = lux = light_sensor.value
    light_label.text = 'Light Sensor: {}'.format(light_value)
    temperature = round(adt.temperature)
    temperature_label.text = 'Temp Sensor: {}'.format(temperature)
    movement_value = movement_sensor.value
    motion_label.text = 'PIR Sensor: {}'.format(movement_value)
```

Click here for more information on string formatting for Python to understand how the .format() function works.

https://adafru.it/Ibh

The button handler

Here is where we decide what happens when the onscreen buttons are pressed. This code will only be run if the screen is touched based on whether `b.contains(touch)` or not.

Button 1 is tested first using `if i=0:` because **Button 1** is the first button in the button group array.

Then if `button1_state == 0` that means that it was off when the button was pressed, so we will now switch the `button1_state` to **1** so that it is **ON**. The opposite is done if `button1_state` started with a value of **1** since the test statement is **FALSE**. This is a simple way to make a toggle state button. We are also using `b.selected = True/False` to change the look of the button when toggled. Last thing for Button 1 is to use `client.publish(mqtt_button1, button1_state )` to publish the new state of Button 1 and then we use `while ts.touch_point:` as a debounce so that nothing happens until the button is released.

Button 2 is tested first using `if i=1:` because **Button 2** is the second button in the button group array.

This is a more simple button and it will just use `client.publish(mqtt_button2, 1)` to publish the Pressed state of the button. It will then wait for the button to be released before it resets Button 2 and calls `client.publish(mqtt_button2, 0)` to publish the Not Pressed state of the button. This will allow us to create Automations later for short and long pressing of this button.

```
# Read display button press
    touch = ts.touch_point
    if touch:
        for i, b in enumerate(buttons):
            if b.contains(touch):
                print('Sending button%d pressed' % i)
                if i == 0:
                    # Toggle switch button type
                    if button1_state == 0:
                        button1_state = 1
                        b.label = "ON"
                        b.selected = False
                        print("Button 1 ON")
                    else:
                        button1_state = 0
                        b.label = "OFF"
                        b.selected = True
                        print("Button 1 OFF")
                    print('Sending button 1 state: ')
                    client.publish(mqtt_button1, button1_state)
                    # for debounce
                    while ts.touch_point:
                        print("Button 1 Pressed")
                if i == 1:
                    # Momentary button type
                    b.selected = True
                    print('Sending button 2 state: ')
                    client.publish(mqtt_button2, 1)
                    # for debounce
                    while ts.touch_point:
                        print("Button 2 Pressed")
                    print("Button 2 reliced")
                    print('Sending button 2 state: ')
                    client.publish(mqtt_button2, 0)
                    b.selected = False
```

Publishing the Sensors

Now we get to the last bit where we simply publish the values of each sensor to it's relevant MQTT topic. This is done by use of the `client.publish()` function which needs the following parameters:

- **MQTT Topic** to publish to
- The **message** to publish in string format

```
# Publish sensor data to MQTT
    print('Sending light sensor value: %d' % light_value)
    client.publish(mqtt_lux, light_value)

    print('Sending temperature value: %d' % temperature)
    client.publish(mqtt_temperature, temperature)

    print('Sending motion sensor value: %d' % movement_value)
    client.publish(mqtt_PIR, '{}'.format(movement_value))
```

And that is the end of our code. If you need more help with getting this code to work, have a look at the following guides that were used to create this code.

Adafruit PyPortal - IoT for CircuitPython

https://adafru.it/Ecp

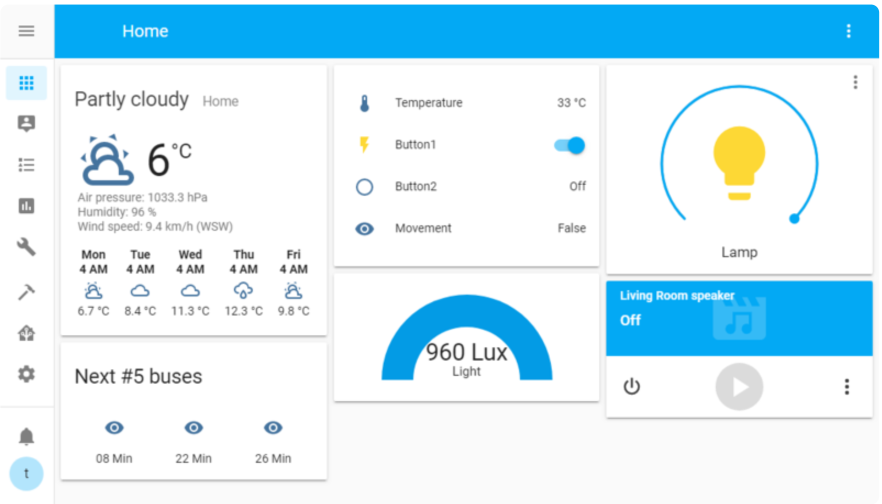
CircuitPython Display Support Using displayio

https://adafru.it/EGh

MQTT in CircuitPython

https://adafru.it/FGp

Home Assistant Configuration



You can use this PyPortal project with any system that can use MQTT. This guide will go into detail on connecting with Home Assistant.

If you are not familiar with Home Assistant then you may want to have a look at the following Learning Guide on setting up your own Home Assistant server.

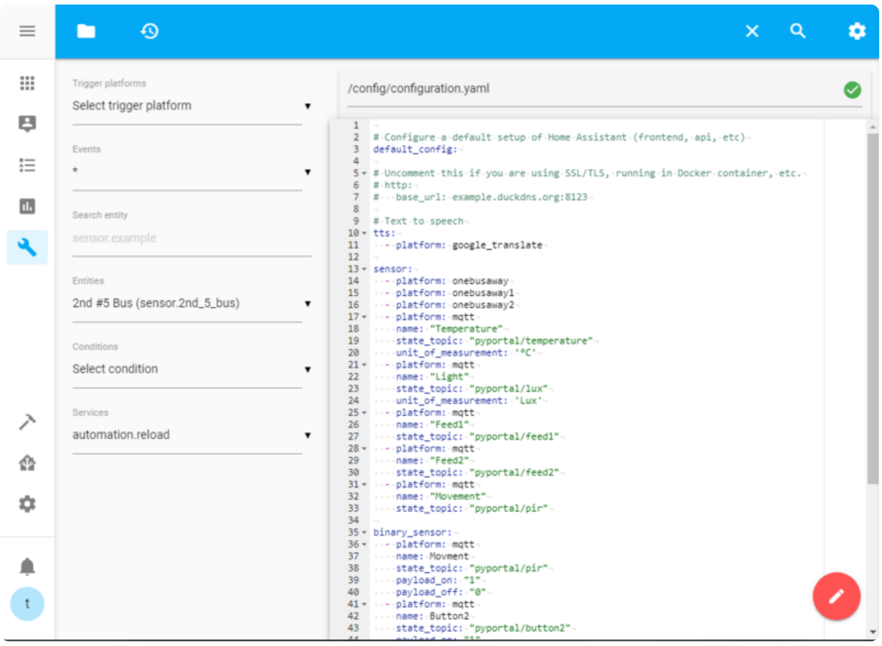
Set up Home Assistant with a Raspberry Pi

https://adafru.it/Ibi

- Next we will set up some Integrations in the Home Assistant Configuration file.
- Find out how to pull data from Home Assistant and send it to the PyPortal via MQTT.
- Then create some Automations to handle the PyPortal on-screen buttons.

Configuration code

Some code will need to be added to the **configuration.yaml** file in Home Assistant. This will set up all the interactions with the sensors on the PyPortal. There are a few ways to do this, and if you are not familiar with Home Assistant, I recommend reading this guide on adding Configurator (https://adafru.it/Ibj) and then this one on editing the **configuration.yaml** file (https://adafru.it/FXM).



Trigger platforms
Select trigger platform

Events
*

Search entity
sensor.example

Entities
2nd #5 Bus (sensor.2nd_5_bus)

Conditions
Select condition

Services
automation.reload

/config/configuration.yaml

```

# Configure a default setup of Home Assistant (frontend, api, etc)
default_config:

# Uncomment this if you are using SSL/TLS, running in Docker container, etc.
# http:
#   base_url: example.duckdns.org:8123

# Text to speech
tts:
  - platform: google_translate

sensor:
  - platform: onebusaway
  - platform: onebusaway1
  - platform: onebusaway2
  - platform: mqtt
    name: "Temperature"
    state_topic: "pyportal/temperature"
    unit_of_measurement: '°C'
  - platform: mqtt
    name: "Light"
    state_topic: "pyportal/lux"
    unit_of_measurement: 'Lux'
  - platform: mqtt
    name: "Feed1"
    state_topic: "pyportal/feed1"
  - platform: mqtt
    name: "Feed2"
    state_topic: "pyportal/feed2"
  - platform: mqtt
    name: "Movement"
    state_topic: "pyportal/pir"

binary_sensor:
  - platform: mqtt
    name: Movment
    state_topic: "pyportal/pir"
    payload_on: "1"
    payload_off: "0"
  - platform: mqtt
    name: Button2
    state_topic: "pyportal/button2"
```

Add the following information onto your **configuration.yaml** file, but be sure to place each sensor type into their respective categories. This is most important if you already have devices listed under **sensor**, **binary_sensor**, or **switch**.

```
sensor:
  - platform: mqtt
    name: "Temperature"
    state_topic: "pyportal/temperature"
    unit_of_measurement: '°C'
  - platform: mqtt
    name: "Light"
    state_topic: "pyportal/lux"
    unit_of_measurement: 'Lux'

binary_sensor:
  - platform: mqtt
    name: Movment
    state_topic: "pyportal/pir"
    payload_on: "1"
    payload_off: "0"
  - platform: mqtt
    name: Button2
    state_topic: "pyportal/button2"
    payload_on: "1"
    payload_off: "0"

switch:
  - platform: mqtt
    name: Button1
    command_topic: "pyportal/button1"
    state_topic: "pyportal/button1"
    payload_on: "1"
    payload_off: "0"
    state_on: "1"
    state_off: "0"
```

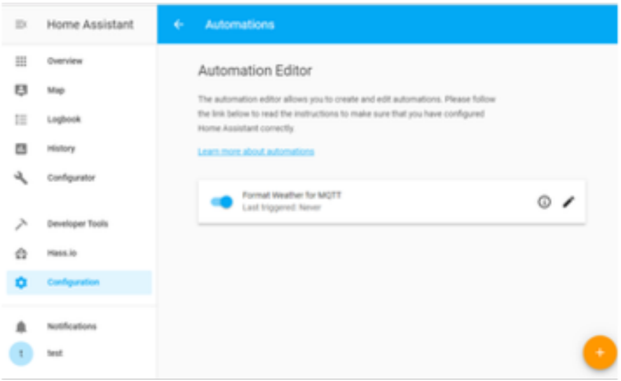
When you have checked your YAML code and saved, go to **Server Control** from the **Configuration** menu and click **RESTART**.

After a few seconds, you will see some text appear that will say **Connection lost. Reconnecting...** at the bottom left of the window. When that text disappears, the server is back online and your changes have been loaded.

Send Data to the PyPortal

Creating the Automation

The first thing we need to do is create an Automation that will pass data from Home Assistant to our MQTT topics whenever that information is updated. To do this you will want to open the main menu on the left side of the Home Assistant screen.



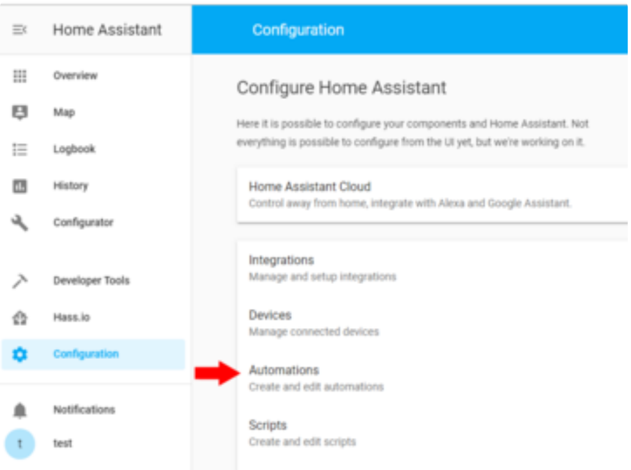
Automation Editor

The automation editor allows you to create and edit automations. Please follow the link below to read the instructions to make sure that you have configured Home Assistant correctly.

Learn more about automations

Format Weather for MQTT
Last triggered: Never

While Home Assistant can use MQTT, most data within that system is handled using an on-board API. So we will need to create an automation that takes data from that API and formats it to publish to our MQTT topic. This is actually rather easy using the Home Assistant UI, and this will show you how to create the content for **pyportal/feed1** and **pyportal/feed2**.



Configure Home Assistant

Here it is possible to configure your components and Home Assistant. Not everything is possible to configure from the UI yet, but we're working on it.

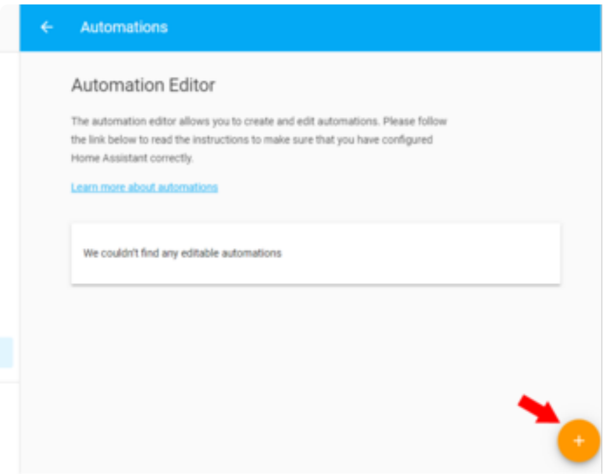
Home Assistant Cloud
Control away from home, integrate with Alexa and Google Assistant.

Integrations
Manage and setup integrations

Devices
Manage connected devices

Automations
Create and edit automations

Scripts
Create and edit scripts

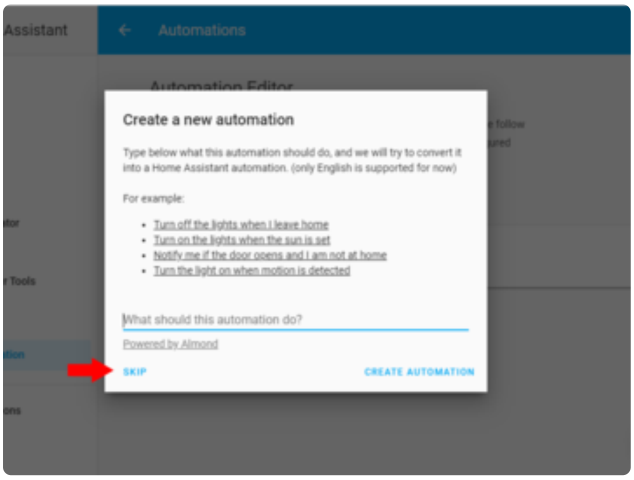


Automation Editor

The automation editor allows you to create and edit automations. Please follow the link below to read the instructions to make sure that you have configured Home Assistant correctly.

Learn more about automations

We couldn't find any editable automations



Create a new automation

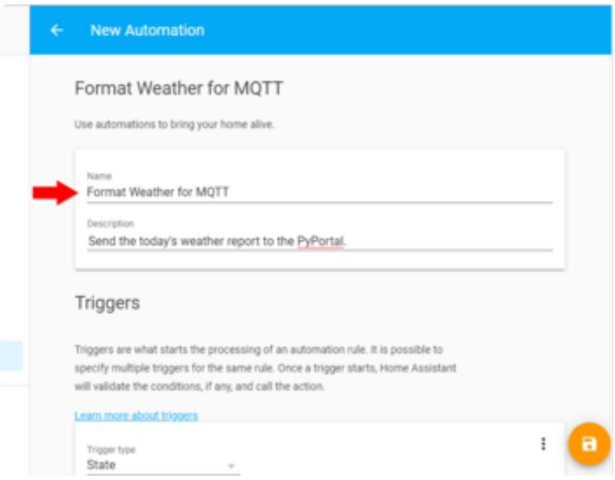
Type below what this automation should do, and we will try to convert it into a Home Assistant automation. (only English is supported for now)

For example:

- Turn off the lights when I leave home
- Turn on the lights when the sun is set
- Notify me if the door opens and I am not at home
- Turn the light on when motion is detected

What should this automation do?

Powered by Almond



Format Weather for MQTT

Use automations to bring your home alive.

Name: Format Weather for MQTT

Description: Send the today's weather report to the PyPortal

Triggers

Triggers are what starts the processing of an automation rule. It is possible to specify multiple triggers for the same rule. Once a trigger starts, Home Assistant will validate the conditions, if any, and call the action.

Learn more about triggers

Trigger type: State

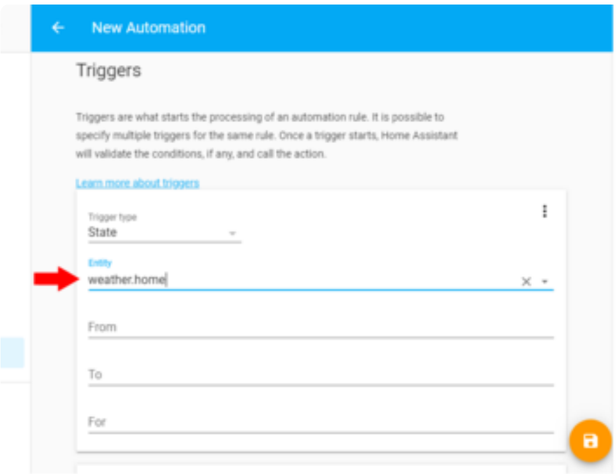
1. Click on the **Configuration** menu
2. Scroll to and click **Automations**
3. Click the + at the bottom right of the screen to create a new Automation.

   Note: to edit an existing Automation, click the pencil icon to the right of the Automation you wish to edit.

4. You may be asked to type out what your automation will do so that the new Home Assistant AI can create the Automation. What we are doing is a bit more complex so you will just click **SKIP** for this option so we can format the Automation normally.
5. Enter a name for your Automation like **Format Weather for MQTT** or something like that.
6. You can also add a Description to explain what this automation is all about.

The Trigger

Now that we have our Automation created, we need to tell it when this automation should be executed. This is called the Trigger and there are many ways we can set it up. For this example we will set our automation to trigger whenever the weather report from Home Assistant changes its State.



Triggers

Triggers are what starts the processing of an automation rule. It is possible to specify multiple triggers for the same rule. Once a trigger starts, Home Assistant will validate the conditions, if any, and call the action.

Trigger type: State

Entity: weather.home

From

To

For

1. Ensure that the **Trigger type** is set to **State**
2. Select the **Entity** option
3. Scroll to and select your main weather entity. This will most likely be **weather.home**

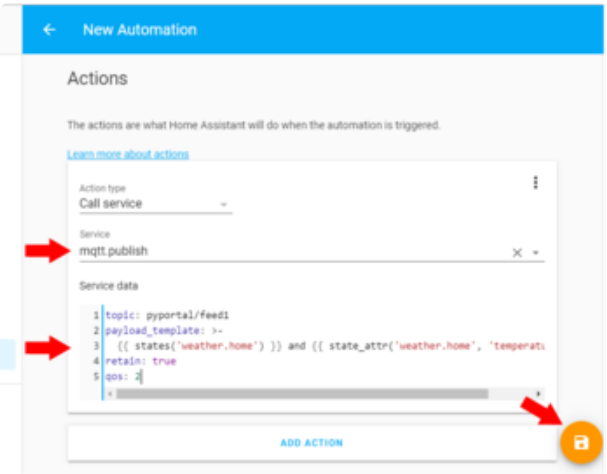
The **From**, **To**, and **For** options are useful if you want this to trigger only if the weather changes from **sunny** to **rainy**, but we want this to trigger whenever there is any change to this entity. So we will leave these blank and any change to the **weather.home** entities **state** will result in this automation running.

You can add as many triggers as you want and if any of them are true, then the automation will run.

Conditions are similar to Triggers but are not required. If you set any Conditions, all of them need to be True for the Automation to be executed. An example of how this could be used would be to make it so that this automation only ran if someone was home and the sun is up.

Actions

Now we get to the fun part, were we tell the automation what to do once triggered. You can have this do as many things as you like, but for this example we will simply format a message for MQTT that contains the current weather and temperature. This process uses Data Templates from Home Assistant so that we can get additional data attributes from the **weather.home** entity.



New Automation

Actions

The actions are what Home Assistant will do when the automation is triggered.

Learn more about actions

Action type: Call service

Service: mqtt.publish

Service data

```
topic: pyportal/feed1
payload_template: >-
  {{ states('weather.home') }} and {{ state_attr('weather.home', 'temperatu
retain: true
qos: 2
```

1. Ensure that **Call service** is selected for **Action Type**.
2. Select the **Service** drop-down.
3. Scroll to and select **mqtt.publish**
4. Now select the **Service data** field.
5. Enter the following YAML code into the **Service data** field:

```
topic: pyportal/feed1
payload_template: &gt;-
  {{ states('weather.home') }} and {{ state_attr('weather.home', 'temperature') }}°
retain: true
qos: 2
```

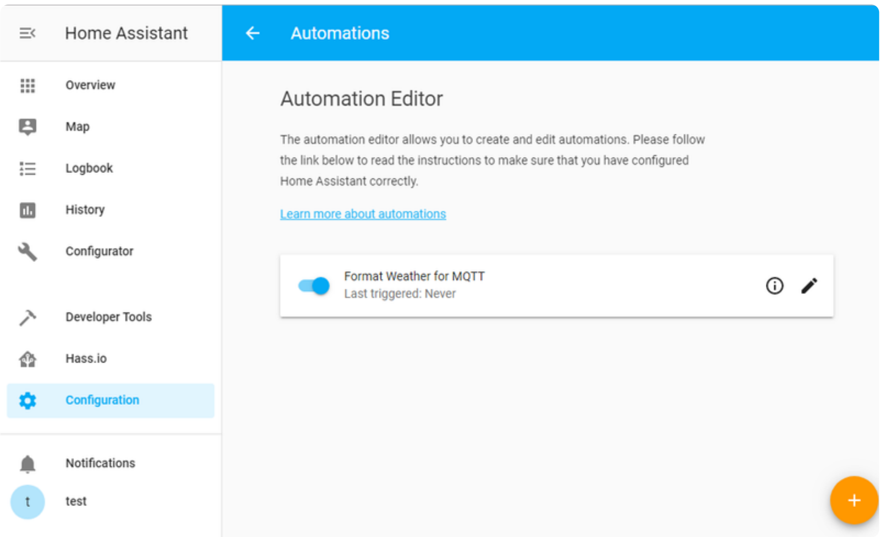
- **Topic:** The MQTT topic that you want to publish to.
- **Retain:** Is this message flagged to be retained.
- **QOS:** Quality of Service for this message. QOS of 2 ensures that the message is delivered no more or less than one time.
- **Payload Template:** This allows you to pull data from the Home Assistant local API and format it into a message string.
  - {{ **states('entity_id')** }} - prints the current State of the stated Entity.
  - {{ **state_attr('entity_id', 'entity_attribute')** }} - prints the value of a particular attribute from a stated Entity.

For more information about data template formatting, click here.

https://adafru.it/Ibk

Now you should be able to click the Save icon at the bottom right of the screen.

You will want to create another one of these for **pyportal/feed2** and any other data that does not already use MQTT before displaying on your PyPortal.



Automation Editor

The automation editor allows you to create and edit automations. Please follow the link below to read the instructions to make sure that you have configured Home Assistant correctly.

Learn more about automations

Format Weather for MQTT
Last triggered: Never

Going Further

If you want to format your message so that it displays information on multiple lines, you can use \n to indicate a new line. Here is an example using data from some sensors pulling data from the OneBusAway REST API:

```
topic: pyportal/feed1
payload_template: |-
  {{ states('sensor.next_bus') }}mins
        Then: {{states('sensor.2nd_bus') }}mins
          and: {{ states('sensor.3rd_bus') }}mins
retain: true
qos: 2
```

For this to work well you would also want to add triggers for each of the sensors, so that data is updated with the latest information for all three sensors.

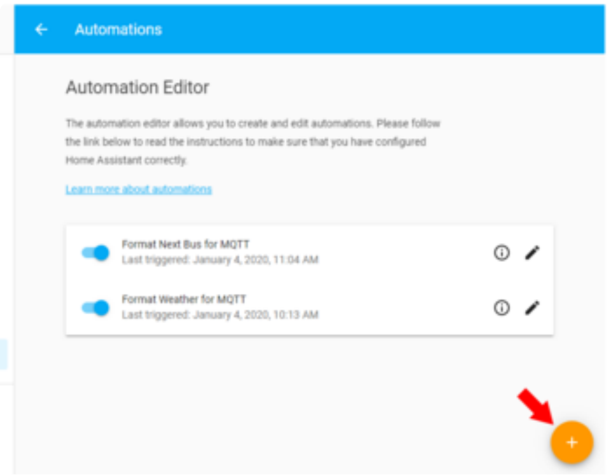
The Display Buttons

While we could just as well have tied the MQTT topics to a device for direct control, I wanted to have the PyPortal use its own topic for the buttons. This will allow us to create an automation that it triggered when the button is pressed. So we get to use the Home Assistant UI rather than having to change the code on the PyPortal if we want other things to happen. Using Automations also allows us to do more than just one thing when we press the button.

For this example, we will be using the buttons to control a LIFX light along with a Google Home Mini. We will also go through using the buttons with conditions to activate one script based on what time of day it is.

Creating the Toggle Button Automations

To handle button1 we will be creating an Automation for turning on and one for switching to off. This will give us more control over what each state will do so that it can be more than just a light switch.



Automations

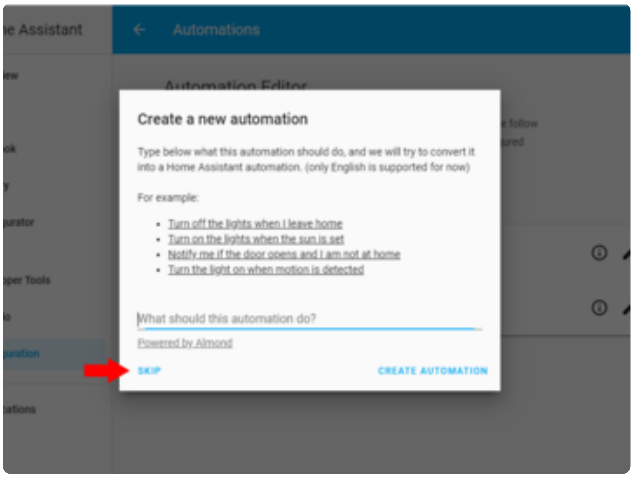
Automation Editor

The automation editor allows you to create and edit automations. Please follow the link below to read the instructions to make sure that you have configured Home Assistant correctly.

Learn more about automations

Format Next Bus for MQTT
Last triggered: January 4, 2020, 11:04 AM

Format Weather for MQTT
Last triggered: January 4, 2020, 10:13 AM



Create a new automation

Type below what this automation should do, and we will try to convert it into a Home Assistant automation. (only English is supported for now)

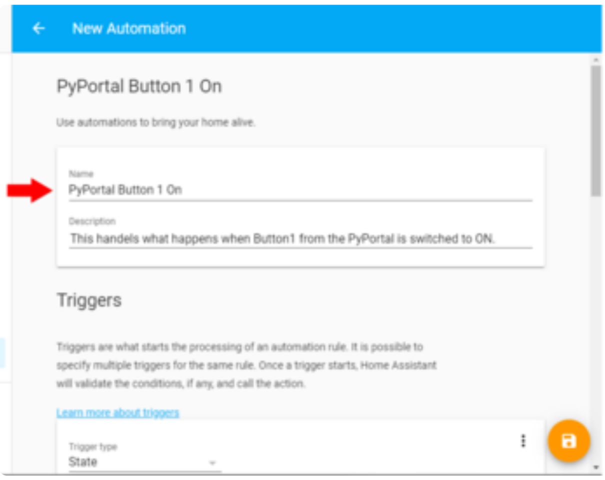
For example:

- Turn off the lights when I leave home
- Turn on the lights when the sun is set
- Notify me if the door opens and I am not at home
- Turn the light on when motion is detected

What should this automation do?

Powered by Almond

SKIP CREATE AUTOMATION



New Automation

PyPortal Button 1 On

Use automations to bring your home alive.

Name
PyPortal Button 1 On

Description
This handels what happens when Button1 from the PyPortal is switched to ON.

Triggers

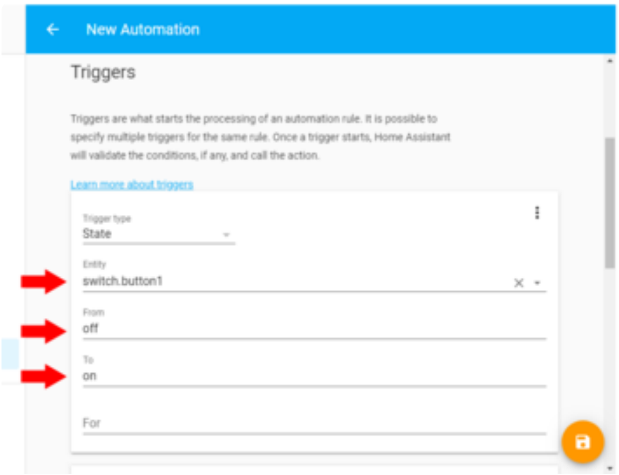
Triggers are what starts the processing of an automation rule. It is possible to specify multiple triggers for the same rule. Once a trigger starts, Home Assistant will validate the conditions, if any, and call the action.

Learn more about triggers

Trigger type
State

1. Click on the **Configuration** menu
2. Scroll to and click **Automations**
3. Click the + at the bottom right of the screen to create a new Automation.

   Note: to edit an existing Automation, click the pencil icon to the right of the Automation you wish to edit.
4. You may be asked to type out what your automation will do so that the new Home Assistant AI can create the Automation. What we are doing is a bit more complex so you will just click **SKIP** for this option so we can format the Automation normally.
5. Enter a name for your Automation like **PyPortal Button 1 On** or something like that.
6. Set the Trigger **Entity** to **switch.button1**
7. Enter **off** into the **From** field
8. Enter **on** into the **To** field
9. Scroll down to **Actions**
10. Select **light.turn_on** for the **Service**
11. Enter the following into the **Service data** field:



Triggers

Triggers are what starts the processing of an automation rule. It is possible to specify multiple triggers for the same rule. Once a trigger starts, Home Assistant will validate the conditions, if any, and call the action.

Learn more about triggers

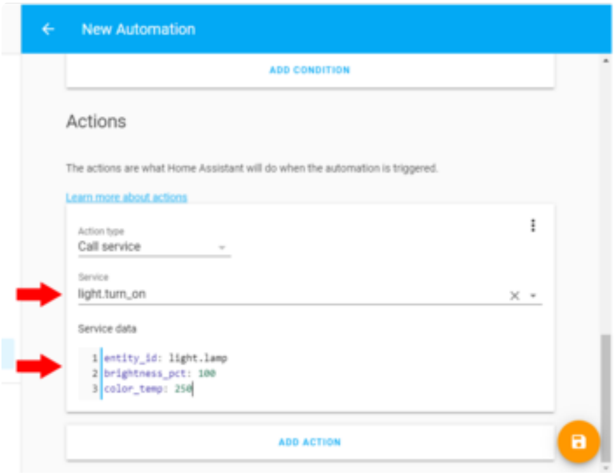
Trigger type: State

Entity: switch.button1

From: off

To: on

For



Actions

The actions are what Home Assistant will do when the automation is triggered.

Learn more about actions

Action type: Call service

Service: light.turn_on

Service data

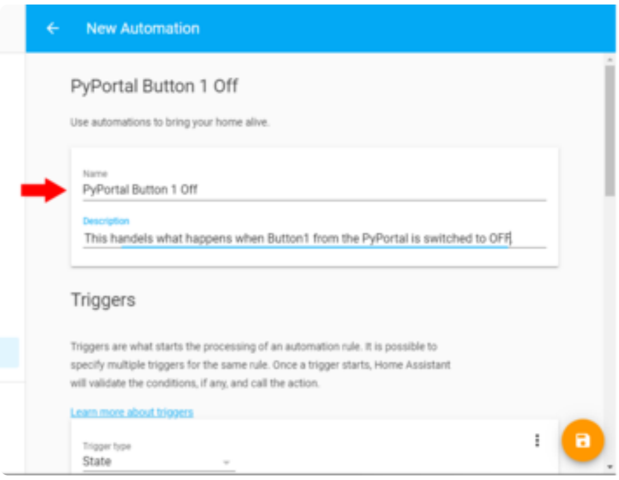
```
entity_id: light.lamp
brightness_pct: 100
color_temp: 250
```

```
entity_id: light.lamp
brightness_pct: 100
color_temp: 250
```

- **entity_id** tells the action what light to turn on.
- **brightness_pct** sets the light level to 100%.
- **color_temp** sets the light to a lukewarm white

When finished, click the **save** icon at the bottom right of the browser.

Now we will create another Automation for **PyPortal Button 1 Off** so that each state of Button1 is covered. So just like before, create a new Automation.



PyPortal Button 1 Off

Use automations to bring your home alive.

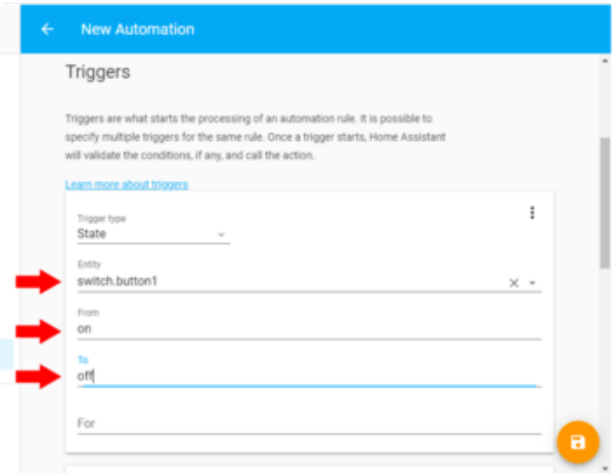
Name: PyPortal Button 1 Off

Description: This handels what happens when Button1 from the PyPortal is switched to OFF

Triggers

Triggers are what starts the processing of an automation rule. It is possible to specify multiple triggers for the same rule. Once a trigger starts, Home Assistant will validate the conditions, if any, and call the action.

Trigger type: State



Triggers

Triggers are what starts the processing of an automation rule. It is possible to specify multiple triggers for the same rule. Once a trigger starts, Home Assistant will validate the conditions, if any, and call the action.

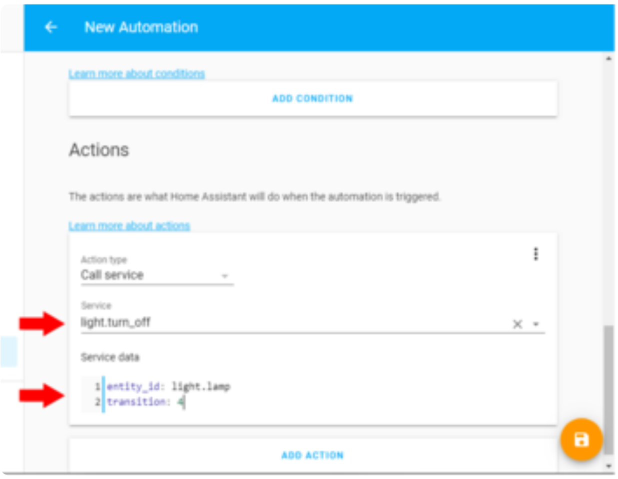
Trigger type: State

Entity: switch.button1

From: on

To: off

For:



Actions

The actions are what Home Assistant will do when the automation is triggered.

Action type: Call service

Service: light.turn_off

Service data:

```
entity_id: light.lamp
transition: 4
```

1. Enter a name for your Automation like **PyPortal Button 1 Off** or something like that.
2. Set the Trigger **Entity** to **switch.button1**
3. Enter **on** into the **From** field
4. Enter **off** into the **To** field
5. Scroll down to **Actions**
6. Select **light.turn_off** for the **Service**
7. Enter the following into the **Service data** field:

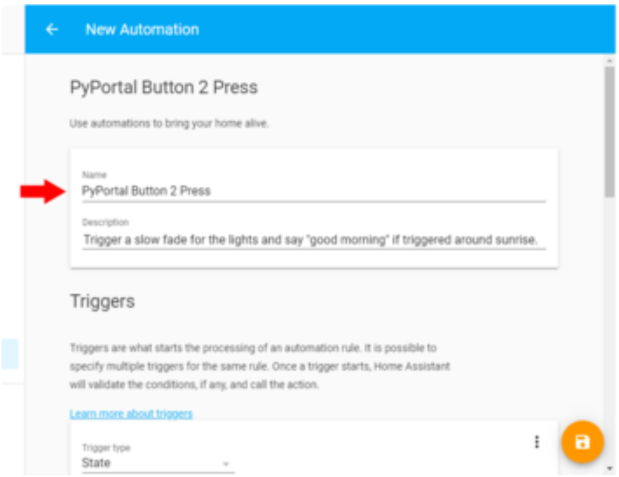
```
entity_id: light.lamp
transition: 4
```

- **entity_id** tells the action what light to turn on.
- **transition** will make it so the light will fade to off in 4 seconds.

When finished, click the **save** icon at the bottom right of the browser.

Automation for Button 2 press

This will handle what happens when the **state** of entity **binary_sensor.button2** changes from **off** to **on**. Once triggered this Automation will fade the light on slowly and say "Good Morning" if pressed before 9AM.



PyPortal Button 2 Press

Use automations to bring your home alive.

Name
PyPortal Button 2 Press

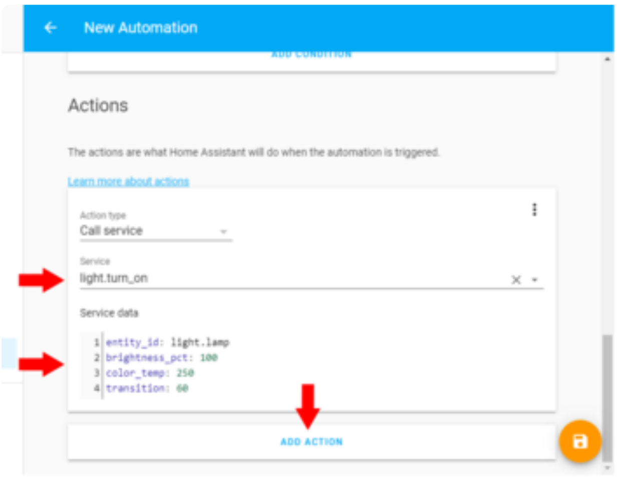
Description
Trigger a slow fade for the lights and say "good morning" if triggered around sunrise.

Triggers

Triggers are what starts the processing of an automation rule. It is possible to specify multiple triggers for the same rule. Once a trigger starts, Home Assistant will validate the conditions, if any, and call the action.

Learn more about triggers

Trigger type
State



Actions

The actions are what Home Assistant will do when the automation is triggered.

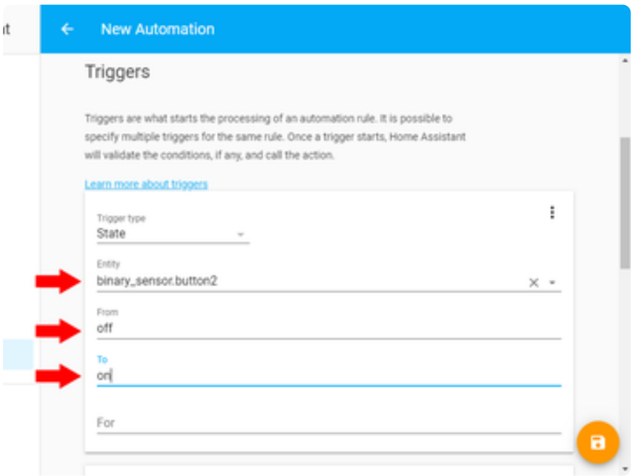
Learn more about actions

Action type
Call service

Service
light.turn_on

Service data

```
entity_id: light.lamp
brightness_pct: 100
color_temp: 250
transition: 60
```



Triggers

Triggers are what starts the processing of an automation rule. It is possible to specify multiple triggers for the same rule. Once a trigger starts, Home Assistant will validate the conditions, if any, and call the action.

Learn more about triggers

Trigger type
State

Entity
binary_sensor.button2

From
off

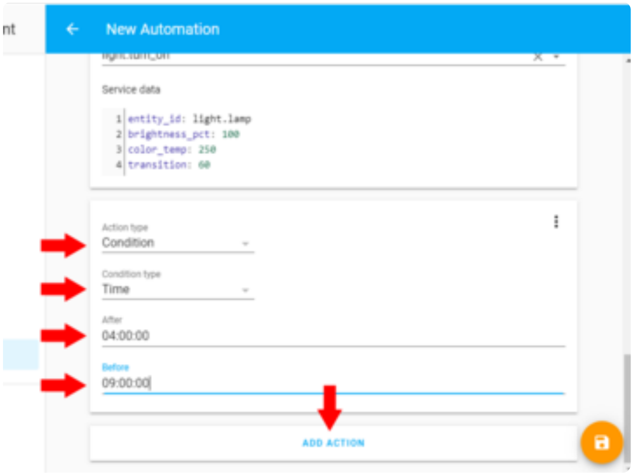
To
on

For

1. Create a new Automation just like before, but name this one something like **PyPortal Button 2 Press**
2. Set the Trigger **Entity** to **binary_sensor.button2**
3. Enter **off** into the **From** field
4. Enter **on** into the **To** field
5. Scroll down to **Actions**
6. Select **light.turn_on** for the **Service**
7. Enter the following into the **Service data** field:

```
entity_id: light.lamp
brightness_pct: 100
color_temp: 250
transition: 60
```

To this we are going to add a condition in the Actions section. If this condition is true, than the following Actions will run or the Automation ends if the condition is false. Basically the light will fade from off to on when Button2 is pressed, but the next action will only take place if the time is between 4am and 9am.



Service data

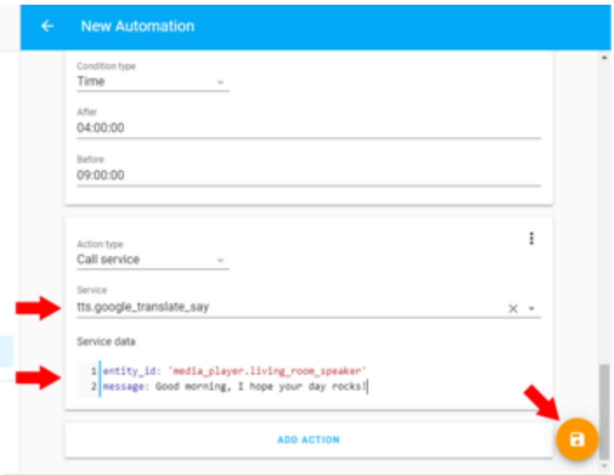
```
entity_id: light.lamp
brightness_pct: 100
color_temp: 250
transition: 60
```

Action type: Condition

Condition type: Time

After: 04:00:00

Before: 09:00:00



Condition type: Time

After: 04:00:00

Before: 09:00:00

Action type: Call service

Service: tts.google_translate_say

Service data

```
entity_id: 'media_player.living_room_speaker'
message: Good morning, I hope your day rocks!
```

1. Click **ADD ACTION** to add a new Action
2. For **Action Type**, select **Condition**
3. For **Condition type**, select **Time**
4. Enter the time you want the condition to start being true in the **After** field.
5. Enter the time you want the condition to end in the **Before** field.

   Note: this system uses 24 hour time in the format HH:MM:SS

6. Add another action by clicking **ADD ACTION**
7. For **Action Type**, keep it on **Call Service**
8. Select **tts.google_translate_say** for **Service**
9. Enter the following into the **Service data** field:

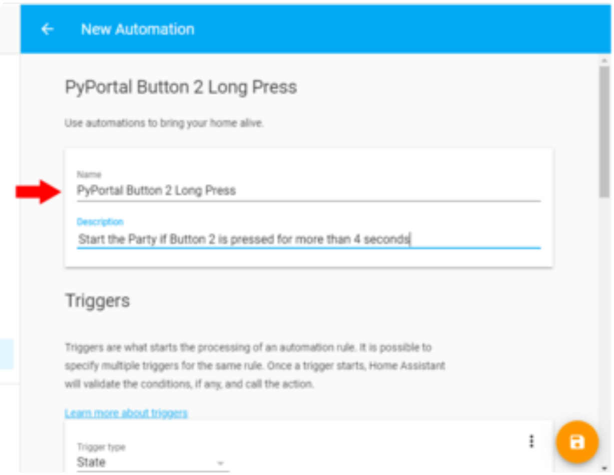
```
entity_id: 'media_player.living_room_speaker'
message: Good morning, I hope your day rocks!
```

So what **tts.google_translate_say** does is to convert the text from our message into sound and then send it to be played on the **Google Home** speaker that is selected by **entity_id**.

When finished, click the **save** icon at the bottom right of the browser.

Automation for Long Press of Button 2

This will handle what happens when the **state** of entity **binary_sensor.button2** changes from **off** to **on** for more than **2 seconds**. If this Automation is triggered, it will start a Party Mode.



New Automation

PyPortal Button 2 Long Press

Use automations to bring your home alive.

Name: PyPortal Button 2 Long Press

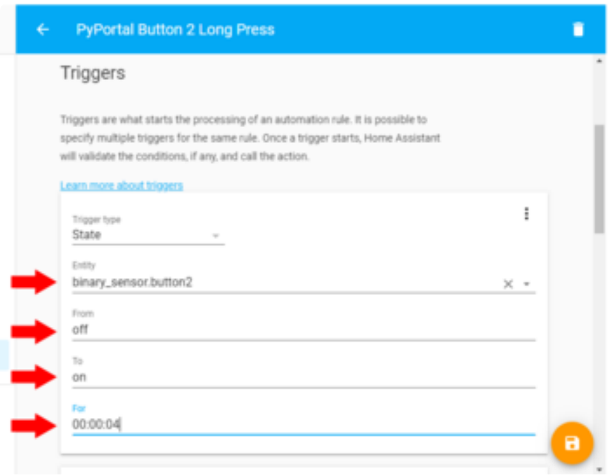
Description: Start the Party if Button 2 is pressed for more than 4 seconds

Triggers

Triggers are what starts the processing of an automation rule. It is possible to specify multiple triggers for the same rule. Once a trigger starts, Home Assistant will validate the conditions, if any, and call the action.

Learn more about triggers

Trigger type: State



PyPortal Button 2 Long Press

Triggers

Triggers are what starts the processing of an automation rule. It is possible to specify multiple triggers for the same rule. Once a trigger starts, Home Assistant will validate the conditions, if any, and call the action.

Learn more about triggers

Trigger type: State

Entity: binary_sensor.button2

From: off

To: on

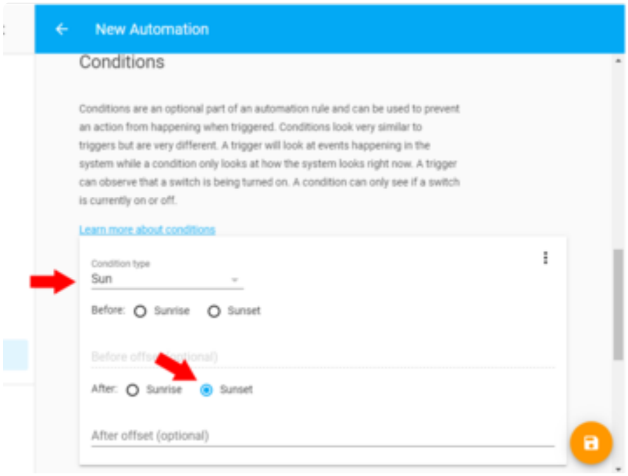
For: 00:00:04

1. Create a new **Automation** just like before, but name this one something like **PyPortal Button 2 Long Press**
2. Set the Trigger **Entity** to **binary_sensor.button2**
3. Enter **off** into the **From** field
4. Enter **on** into the **To** field
5. Enter **00:00:04** into the **For** field

   Note: this uses the time format HH:MM:SS

Conditions

For this Automation, we will be adding a Condition that will need to be TRUE before the Automation can be triggered. For this example, we only want to access Party Mode if the sun has gone down.



New Automation

Conditions

Conditions are an optional part of an automation rule and can be used to prevent an action from happening when triggered. Conditions look very similar to triggers but are very different. A trigger will look at events happening in the system while a condition only looks at how the system looks right now. A trigger can observe that a switch is being turned on. A condition can only see if a switch is currently on or off.

Learn more about conditions

Condition type: Sun

Before: ○ Sunrise ○ Sunset

Before offset (optional)

After: ○ Sunrise ● Sunset

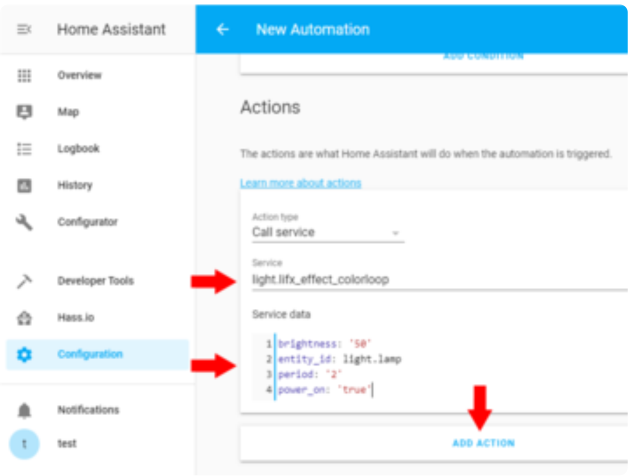
After offset (optional)

1. Scroll to and click on **ADD CONDITION**
2. For **Condition type**, select **Sun**
3. Click **Sunset** next to the **After** section

Now our Automation can only be triggered if it is after the Sun has gone down.

Add the Actions

Now we are going to add our actions to start Party Mode. This will include setting the LIFX light to cycle through colors and have the Google Home speaker play music from an internet radio station.



Actions

The actions are what Home Assistant will do when the automation is triggered.

Learn more about actions

Action type: Call service

Service: light.lifx_effect_colorloop

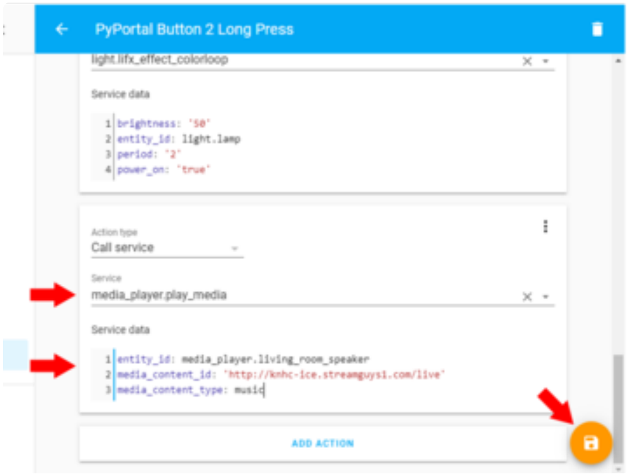
Service data:

```
brightness: '50'
entity_id: light.lamp
period: '2'
power_on: 'true'
```

1. Scroll down to **Actions**
2. For **Service**, select **light.lifx_effect_colorloop**
3. Enter the following into the **Service data** field:

```
brightness: '50'
entity_id: light.lamp
period: '2'
power_on: 'true'
```

The **light.lifx_effect_colorloop** is a service that is made just for LIFX bulbs though there is a similar one for HUE lights as well. If you are using a generic light, you can use the service **light.on** and **effect: colorloop** for the **Service data**.



light.lifx_effect_colorloop

Service data:

```
brightness: '50'
entity_id: light.lamp
period: '2'
power_on: 'true'
```

Action type: Call service

Service: media_player.play_media

Service data:

```
entity_id: media_player.living_room_speaker
media_content_id: 'http://knhc-ice.streamguys1.com/live'
media_content_type: music
```

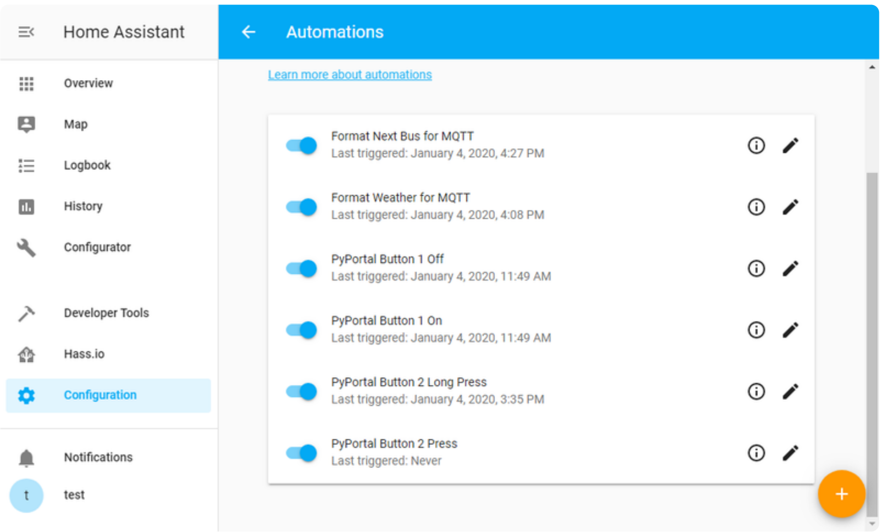
Now we will get our Google Home speaker to play music.

1. Add another action by clicking **ADD ACTION**
2. Select **media_player.play_media** for **Service**
3. Enter the following into the **Service data** field:

```
media_content_id: 'http://knhc-ice.streamguys1.com/live'
entity_id: media_player.living_room_speaker
media_content_type: music
```

The **media_player.play_media** lets you send media files to a connected media device like our Google Home speaker. For this example we are using the internet radio link for Seattle's student run radio station C89.5 as the media source, but you can link this to a music file or any other streaming music link.

When finished, click the **save** icon at the bottom right of the browser.



Format Next Bus for MQTT
Last triggered: January 4, 2020, 4:27 PM

Format Weather for MQTT
Last triggered: January 4, 2020, 4:08 PM

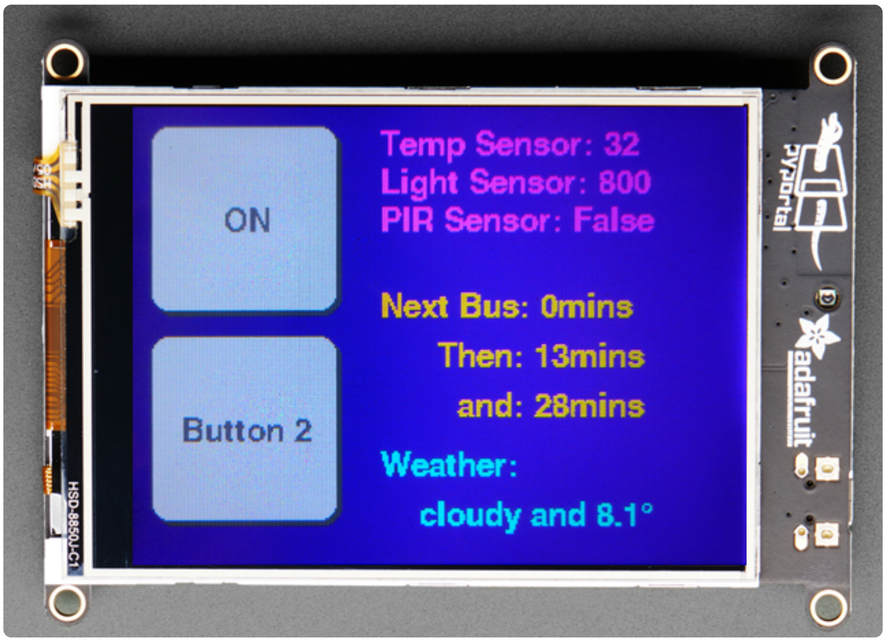
PyPortal Button 1 Off
Last triggered: January 4, 2020, 11:49 AM

PyPortal Button 1 On
Last triggered: January 4, 2020, 11:49 AM

PyPortal Button 2 Long Press
Last triggered: January 4, 2020, 3:35 PM

PyPortal Button 2 Press
Last triggered: Never

Usage



Now that everything is set up your PyPortal just needs power from the USB port and it should connect to your WiFi and Home Assistant.

To turn the Light we connected ON or OFF, just touch the top button and it will switch states.

If the light is off and you want to slowly fade the light on, touch Button 2 until it turns green and let go. This will fade the light from off to on over the course of 60 seconds.

If this button is pressed between the hours of 4 AM and 9 AM, Google Home will say "Good morning".

If you press and hold Button 2 for more than 4 seconds after the sun goes down, Home Assistant will start Party Mode.

Sensor data from the PyPortals Temperature, Light, and PIR sensors are now being sent to Home Assistant and can be used to create more Automations.

Information is also being passed from Home Assistant sensors to the PyPortal and displayed as Feed1 and Feed2