PyPortal Alarm Clock

Created by Dave Astels

https://learn.adafruit.com/pyportal-alarm-clock

Last updated on 2024-03-08 03:23:13 PM EST

Table of Contents

Overview 5
- Parts

Use 8

Hardware 10

Graphics 11

Code 12
- Getting Familiar
- Download Library Files
- The Code

Setup 23
- Imports
- Configuration and Variables
- Logging
- Support Functions

State Machines 26
- The Machine
- Details

The Time State 29
- __init__
- tick
- touch
- enter
- exit

The Mugsy State 35

The Alarm State 36
- __init__
- tick
- touch
- enter
- exit

The Setting State 38
- __init__
- touch
- enter

3D Printing a Case 40
- 3D Printed Parts
- Slice Settings
- Flat Head Screws
- Display Bracket Assembly
- PowerBoost Connections

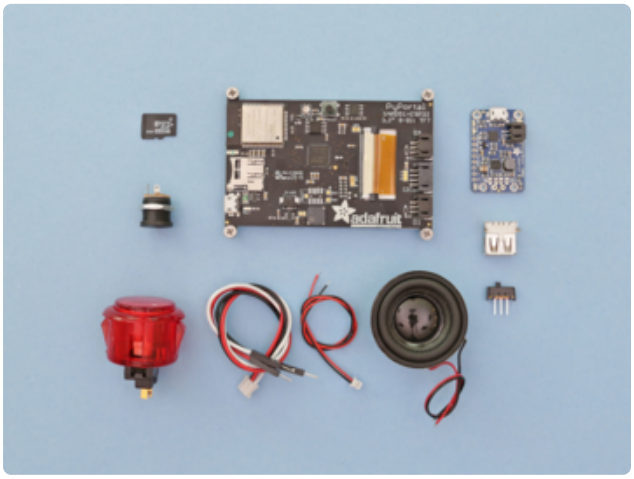
- Mount Display
- Mount PowerBoost
- Arcade Button Holder
- Speaker
- Barrel + Slide Switch
-
- Battery
- Mount Key

An Ongoing Mission... 49

Overview

The PyPortal is an exciting new product that makes it easy to build IoT devices that communicate over WiFi with builtin TLS/SSl support for secure communication. It also has a 3.2" TFT touchscreen, MicroSD card slot, speaker, light sensor, temperature sensor, 8MB of SPI flash, and STEMMA connectors for I2C, and digital/analog connections.

Running the show is an ATSAMD51J20. This is much like the MCU on the other "M4" boards but, like the Metro M4 Grand Central, with 1M flash and 256K of RAM. This provides space for all kinds of CircuitPython code. And because the board contains an ESP32 coprocessor to handle WiFi communication, the SAMD51 can focus on running your code.



In this guide we'll take a PyPortal and a few other pieces and build an alarm clock, complete with a display of the current weather, a snooze button, and the ability to trigger another function... for example asking an IoT coffee maker to start brewing that crucial first coffee of the morning.

The complete code is available, written in CircuitPython. It is easy to make customizations by using any editor to change the code, then copy it back onto the device.

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

Speaker - 40mm Diameter - 4 Ohm 3 Watt

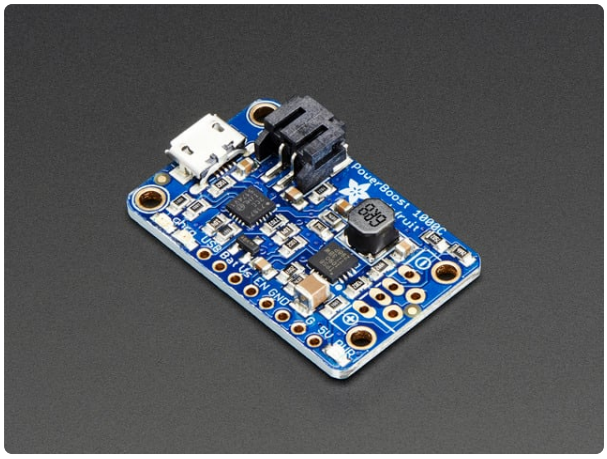
Hear the good news! This speaker is a great addition to any audio project where you need a 4 Ohm impedance and 3W or less of power.At 40mm diameter it...

https://www.adafruit.com/product/3968

Arcade Button - 30mm Translucent Clear

A button is a button, and a switch is a switch, but these translucent arcade buttons are in a class of their own. They're the same size as common arcade controls (often referred to...

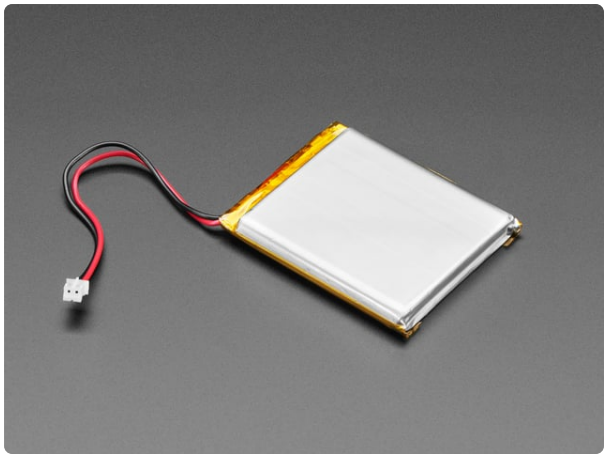
https://www.adafruit.com/product/471



PowerBoost 1000 Charger - Rechargeable 5V Lipo USB Boost @ 1A

PowerBoost 1000C is the perfect power supply for your portable project! With a built-in load-sharing battery charger circuit, you'll be able to keep your power-hungry...

https://www.adafruit.com/product/2465



Lithium Ion Polymer Battery - 3.7v 2500mAh

Lithium-ion polymer (also known as 'lipo' or 'lipoly') batteries are thin, light, and powerful. The output ranges from 4.2V when completely charged to 3.7V. This...

https://www.adafruit.com/product/328

Power related alternatives.

1 x SPDT Slide Switch — https://www.adafruit.com/product/805

Connected to the PowerBoost's enable input.

---

1 x 5V 2A (2000mA) switching power supply — https://www.adafruit.com/product/276

Power supply to use with the barrel connector

---

1 x Panel Mount 2.1mm DC barrel jack — https://www.adafruit.com/product/610

Power Jack

For powering via the micro USB jack

1 x 5V 2.5A Switching Power Supply — https://www.adafruit.com/product/1995

For powering via the micro USB jack

Various hardware bits and pieces for assembly.

6 x M3x5mm bolt — https://www.albanycountyfasteners.com/3-MM-x-5-Phillips-Flat-Head-Machine-Screw-p/1011-1006.htm

For assembly

---

4 x M2.5x5mm bolt — https://www.albanycountyfasteners.com/3-MM-x-5-Phillips-Flat-Head-Machine-Screw-p/1011-1006.htm

For assembly.

---

6 x M2x5mm bolt — https://www.albanycountyfasteners.com/2MM-x-4-Phillips-Flat-Head-Machine-Screw-p/1011-1000.htm

For assembly

---

1 x 3 pin STEMMA cable — https://www.adafruit.com/product/3893

For connecting power

---

1 x Pico blade 2 pin connector/cable — https://www.adafruit.com/product/3922

For connecting the speaker

---

1 x Jumper wires — https://www.adafruit.com/product/758

For generally connections

Use

The clock displays both the time and weather. When the alarm is enabled and the set time is reached, the alarm sounds and the alarm screen is displayed. The alarm sound is taken in the file **alarm.wav** in the **CIRCUITPY** directory.

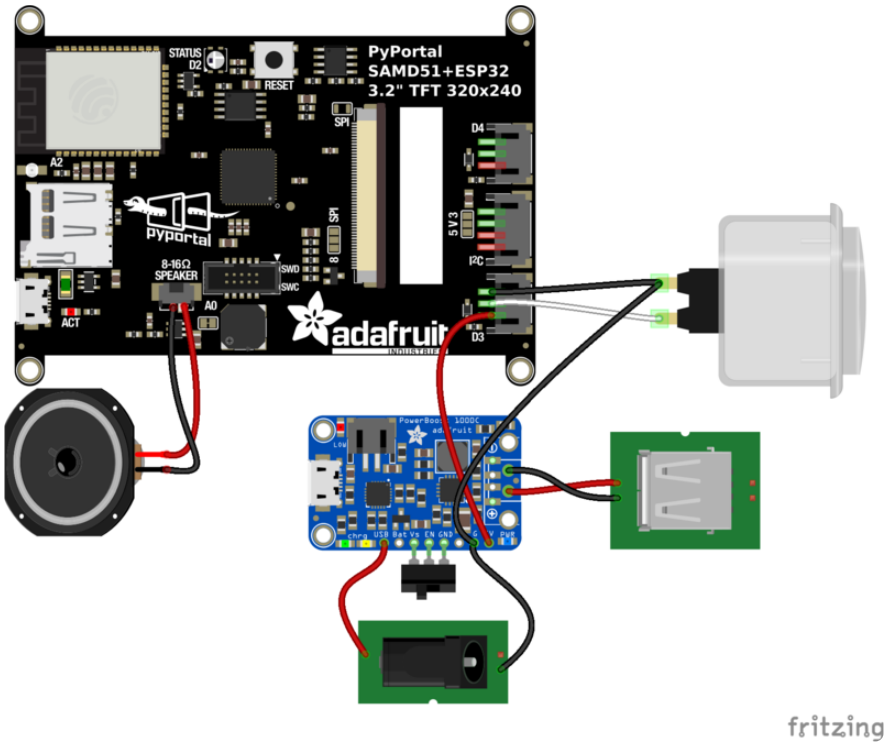
If the screen is touched anywhere, the alarm is silenced until the next day. If the snooze button is pressed instead, the alarm is silenced for 10 minutes. In both cases, the main time screen is displayed. If snoozing is active, an indicator is displayed on the time screen and if the snooze button is pressed, snoozing is canceled. Snoozing is also canceled if you switch to the alarm settings screen.

If you press the alarm button on the time screen (at the top left, next to the weather icon) it switches to the alarm setting screen. There are three touch areas on the left and the alarm time displayed in a large font. On this screen you can enable (by touching the green area) and disable (the red area) as well as adjust the alarm time. You do this by slowly swiping up or down over the hours and minutes.

When you're done, touch the yellow arrow area between the green and red areas. This returns you to the main time screen.

The final piece of functionality is what the author has done as a Mugsy command. The intent with this is to use it to send a command to another IoT device; in this case a Mugsy coffee maker, but it could as easily be to tell your IoT lighting to turn on, or the motorized blinds to open.

Hardware



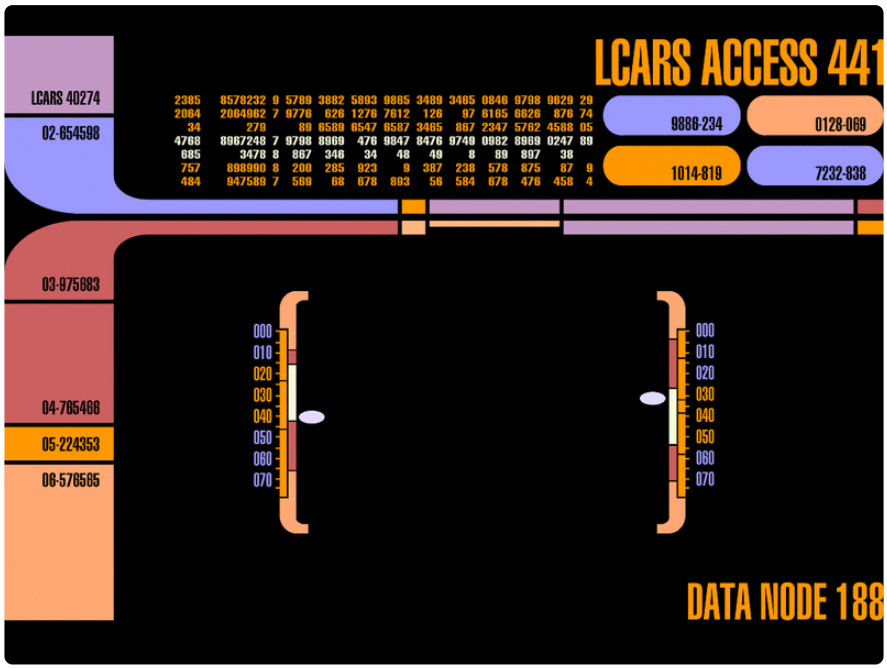
The wiring diagram is shown above. The core of the project is the PyPortal. To that we connect a speaker for more volume, and a big switch for a whackable snooze button. For power, you may use a micro-USB wall supply (in the product list) or do what's shown above.

Using a PowerBoost provides a battery backup to maintain the alarm time in the case of a power outage. Instead of a micro-USB power cord, you can (if desired) add a DC barrel jack and power with a matching 5V wall-wart. This also allows the optional USB-A connector on the PowerBoost to be used directly rather than requiring an additional breakout. The PowerBoost then powers the PyPortal through the 5v connection on the I/O connector. The Power Boost comes with a USB-A connector which can be used for a phone changer. Handy for those (like the author) who are used to charging their phone on their bedside table.

You could forego the entire PowerBoost/battery combo and power the PyPortal directly using a micro-USB power source.

If you use the external speaker, remember to cut the small jumper trace between the on-board speaker and the speaker connector. It's labelled "A0" in the image above.

Graphics



While you can do anything you want with screen backgrounds and such to make it your own, the author went with a Star Trek theme. Have fun with it.

For icons, I found some good ones at Icon8 which distributes them under the license called Creative Commons Attribution-NoDerivs 3.0 Unported (https://adafru.it/EcA). You can find a wealth of free to use (with credit/link given) at https://icons8.com (https://adafru.it/EcB). The alarm button icon is from there.

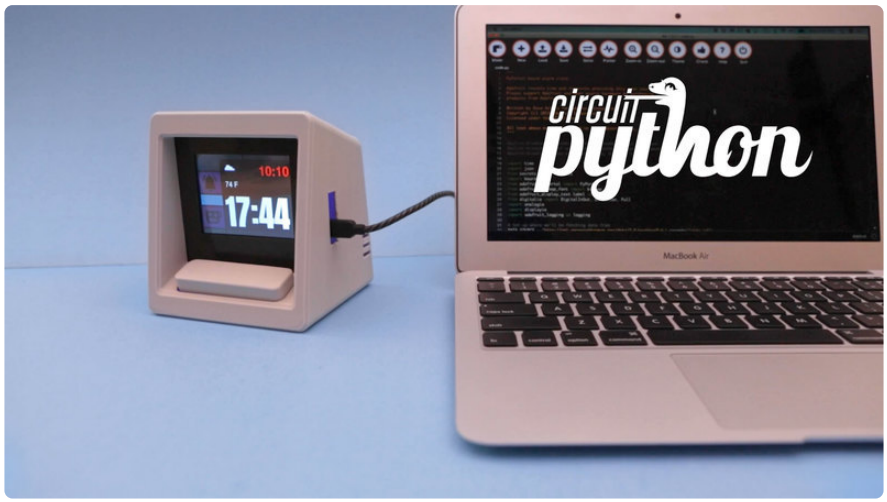
The Mugsy icon is from the Mugsy site (https://adafru.it/EcC).

The LCARS style screens are inspired by Star Trek computer interfaces from The Next Generation onward.

The alarm screen can be any 320x240 24-bit bmp file you like. I found a red alert screen that went with the Star Trek theme.

Weather icons are from the OpenWeatherMap site (https://adafru.it/EcD) which is where weather data is pulled from. The were tweaked slightly by adding a black background for use on the PyPortal.

Code



Getting Familiar

CircuitPython is a programming language based on Python, one of the fastest growing programming languages in the world. It is specifically designed to simplify experimenting and learning to code on low-cost microcontroller boards. Here are some guides which cover the basics:

- Welcome to CircuitPython! (https://adafru.it/cpy-welcome)
- Adafruit PyPortal (https://adafru.it/Ecp)

Be sure you have the latest CircuitPython for PyPortal loaded onto your board. This should be from no earlier than the end of Feb 2019. [At this writing 4.0 Beta 3 is recommended]

CircuitPython is easiest to use within the Mu Editor. If you haven't previously used Mu, this guide will get you started (https://adafru.it/ANO).

Download Library Files

Plug your Feather M4 Express board into your computer via a USB cable. Please be sure the cable is a good power+data cable so the computer can talk to the Feather board.

A new disk should appear in your computer's file explorer/finder called **CIRCUITPY**. This is the place we'll copy the code and code library. If you can only get a drive named **CPLAYBOOT**, load CircuitPython per the PyPortal guide above.

Create a new directory on the **CIRCUITPY** drive named **lib**.

Download the latest CircuitPython driver package to your computer using the green button below. **Match the library you get to the version of CircuitPython you are using**. Save to your computer's hard drive where you can find it.

Download the latest version of CircuitPython for this board via CircuitPython.org

https://adafru.it/Egk

With your computer's file explorer/finder, browse to the bundle and open it up. You'll need to copy three libraries to **CIRCUITPY/lib**:

- the **adafruit_bitmap_font** directory
- the **adafruit_bus_device** directory
- the **adafruit_display_text** directory
- the **adafruit_esp32spi** directory
- **adafruit_pyportal.mpy**

- adafruit_touchscreen.mpy
- neopixel.mpy
- adafruit_logging.mpy

The Code

The main code file is below, but you will need to download the entire repo using the Project Zip link above the file.

```
# SPDX-FileCopyrightText: 2019 Dave Astels for Adafruit Industries
#
# SPDX-License-Identifier: MIT

"""
PyPortal based alarm clock.

Adafruit invests time and resources providing this open source code.
Please support Adafruit and open source hardware by purchasing
products from Adafruit!

Written by Dave Astels for Adafruit Industries
Copyright (c) 2019 Adafruit Industries
Licensed under the MIT license.

All text above must be included in any redistribution.
"""

#pylint:disable=redefined-outer-name,no-member,global-statement
#pylint:disable=no-self-use,too-many-branches,too-many-statements
#pylint:disable=useless-super-delegation, too-many-locals

import time
import json
from secrets import secrets
import board
from adafruit_pyportal import PyPortal
from adafruit_bitmap_font import bitmap_font
from adafruit_display_text.label import Label
from digitalio import DigitalInOut, Direction, Pull
import analogio
import displayio
import adafruit_logging as logging

# Set up where we'll be fetching data from
DATA_SOURCE = 'http://api.openweathermap.org/data/2.5/weather?
id='+secrets['city_id']
DATA_SOURCE += '&appid='+secrets['openweather_token']
# You'll need to get a token from openweather.org, looks like
'b6907d289e10d714a6e88b30761fae22'
DATA_LOCATION = []

####################
# setup hardware

pyportal = PyPortal(url=DATA_SOURCE,
                    json_path=DATA_LOCATION,
                    status_neopixel=board.NEOPIXEL)

light = analogio.AnalogIn(board.LIGHT)

snooze_button = DigitalInOut(board.D3)
snooze_button.direction = Direction.INPUT
```

```
snooze_button.pull = Pull.UP

####################
# variables

# alarm support

alarm_background = 'red_alert.bmp'
alarm_file = 'alarm.wav'
alarm_enabled = True
alarm_armed = True
alarm_interval = 10.0
alarm_hour = 9
alarm_minute = 45
snooze_time = None
snooze_interval = 600.0

# mugsy support
mugsy_background = 'mugsy_background.bmp'

# weather support

icon_file = None
icon_sprite = None
celcius = secrets['celcius']

# display/data refresh timers

refresh_time = None
update_time = None
weather_refresh = None

# The most recently fetched time
current_time = None

# track whether we're in low light mode

low_light = False


####################
# Load the fonts

time_font = bitmap_font.load_font('/fonts/Anton-Regular-104.bdf')
time_font.load_glyphs(b'0123456789:') # pre-load glyphs for fast printing

alarm_font = bitmap_font.load_font('/fonts/Helvetica-Bold-36.bdf')
alarm_font.load_glyphs(b'0123456789:')

temperature_font = bitmap_font.load_font('/fonts/Arial-16.bdf')
temperature_font.load_glyphs(b'0123456789CF')

####################
# Set up logging

logger = logging.getLogger('alarm_clock')
logger.addHandler(logging.StreamHandler())
logger.setLevel(logging.ERROR)            # change as desired

####################
# Functions

def create_text_areas(configs):
    """Given a list of area specifications, create and return test areas."""
    text_areas = []
    for cfg in configs:
        textarea = Label(cfg['font'], text=' '*cfg['size'])
        textarea.x = cfg['x']
        textarea.y = cfg['y']
```

```
        textarea.color = cfg['color']
        text_areas.append(textarea)
    return text_areas


def clear_splash():
    for _ in range(len(pyportal.splash) - 1):
        pyportal.splash.pop()


def touch_in_button(t, b):
    in_horizontal = b['left'] <= t[0] <= b['right']
    in_vertical = b['top'] <= t[1] <= b['bottom']
    return in_horizontal and in_vertical


touched = False

####################
# states

class State(object):
    """State abstract base class"""

    def __init__(self):
        pass


    @property
    def name(self):
        """Return the name of teh state"""
        return ''


    def tick(self, now):
        """Handle a tick: one pass through the main loop"""
        pass


    #pylint:disable=unused-argument
    def touch(self, t, touched):
        """Handle a touch event.
        :param (x, y, z) - t: the touch location/strength"""
        return bool(t)


    def enter(self):
        """Just after the state is entered."""
        pass


    def exit(self):
        """Just before the state exits."""
        clear_splash()


class Time_State(State):
    """This state manages the primary time display screen/mode"""

    def __init__(self):
        super().__init__()
        self.background_day = 'main_background_day.bmp'
        self.background_night = 'main_background_night.bmp'
        self.refresh_time = None
        self.update_time = None
        self.weather_refresh = None
        text_area_configs = [dict(x=88, y=170, size=5, color=0xFFFFFF,
font=time_font),
                             dict(x=210, y=50, size=5, color=0xFF0000,
```

```
font=alarm_font),
                            dict(x=88, y=90, size=6, color=0xFFFFFF,
font=temperature_font)]
        self.text_areas = create_text_areas(text_area_configs)
        self.weather_icon = displayio.Group()
        self.weather_icon.x = 88
        self.weather_icon.y = 20
        self.icon_file = None

        self.snooze_icon = displayio.Group()
        self.snooze_icon.x = 260
        self.snooze_icon.y = 70
        self.snooze_file = None

        # each button has it's edges as well as the state to transition to when
touched
        self.buttons = [dict(left=0, top=50, right=80, bottom=120,
next_state='settings'),
                        dict(left=0, top=155, right=80, bottom=220,
next_state='mugsy')]


    @property
    def name(self):
        return 'time'


    def adjust_backlight_based_on_light(self, force=False):
        """Check light level. Adjust the backlight and background image if it's
dark."""
        global low_light
        if light.value <= 1000 and (force or not low_light):
            pyportal.set_backlight(0.01)
            pyportal.set_background(self.background_night)
            low_light = True
        elif force or (light.value >= 2000 and low_light):
            pyportal.set_backlight(1.00)
            pyportal.set_background(self.background_day)
            low_light = False


    def tick(self, now):
        global alarm_armed, snooze_time, update_time, current_time

        # is the snooze button pushed? Cancel the snooze if so.
        if not snooze_button.value:
            if snooze_time:
                self.snooze_icon.pop()
            snooze_time = None
            alarm_armed = False

        # is snooze active and the snooze time has passed? Transition to alram is
so.
        if snooze_time and ((now - snooze_time) >= snooze_interval):
            change_to_state('alarm')
            return

        # check light level and adjust background & backlight
        #self.adjust_backlight_based_on_light()

        # only query the online time once per hour (and on first run)
        if (not self.refresh_time) or ((now - self.refresh_time) > 3600):
            logger.debug('Fetching time')
            try:
                pyportal.get_local_time(location=secrets['timezone'])
                self.refresh_time = now
            except RuntimeError as e:
                self.refresh_time = now - 3000   # delay 10 minutes before retrying
                logger.error('Some error occured, retrying! - %s', str(e))
```

```
        # only query the weather every 10 minutes (and on first run)
        if (not self.weather_refresh) or (now - self.weather_refresh) > 600:
            logger.debug('Fetching weather')
            try:
                value = pyportal.fetch()
                weather = json.loads(value)

                # set the icon/background
                weather_icon_name = weather['weather'][0]['icon']
                try:
                    self.weather_icon.pop()
                except IndexError:
                    pass
                filename = "/icons/"+weather_icon_name+".bmp"

                if filename:
                    # CircuitPython 6 & 7 compatible
                    if self.icon_file:
                        self.icon_file.close()
                    self.icon_file = open(filename, "rb")
                    icon = displayio.OnDiskBitmap(self.icon_file)

                    icon_sprite = displayio.TileGrid(icon,
                                                     pixel_shader=getattr(icon,
'pixel_shader', displayio.ColorConverter()),
                                                     x=0, y=0)

                    # # CircuitPython 7+ compatible
                    # icon = displayio.OnDiskBitmap(filename)
                    # icon_sprite = displayio.TileGrid(icon,
pixel_shader=icon.pixel_shader)

                    self.weather_icon.append(icon_sprite)

                temperature = weather['main']['temp'] - 273.15 # its...in kelvin
                if celcius:
                    temperature_text = '%3d C' % round(temperature)
                else:
                    temperature_text = '%3d F' % round(((temperature * 9 / 5) + 32))
                self.text_areas[2].text = temperature_text
                self.weather_refresh = now
                try:
                    board.DISPLAY.refresh(target_frames_per_second=60)
                except AttributeError:
                    board.DISPLAY.refresh_soon()
                    board.DISPLAY.wait_for_frame()


            except RuntimeError as e:
                self.weather_refresh = now - 540   # delay a minute before retrying
                logger.error("Some error occured, retrying! - %s", str(e))

        if (not update_time) or ((now - update_time) > 30):
            # Update the time
            update_time = now
            current_time = time.localtime()
            time_string = '%02d:%02d' % (current_time.tm_hour,current_time.tm_min)
            self.text_areas[0].text = time_string
            try:
                board.DISPLAY.refresh(target_frames_per_second=60)
            except AttributeError:
                board.DISPLAY.refresh_soon()
                board.DISPLAY.wait_for_frame()


            # Check if alarm should sound
        if current_time is not None and not snooze_time:
            minutes_now = current_time.tm_hour * 60 + current_time.tm_min
```

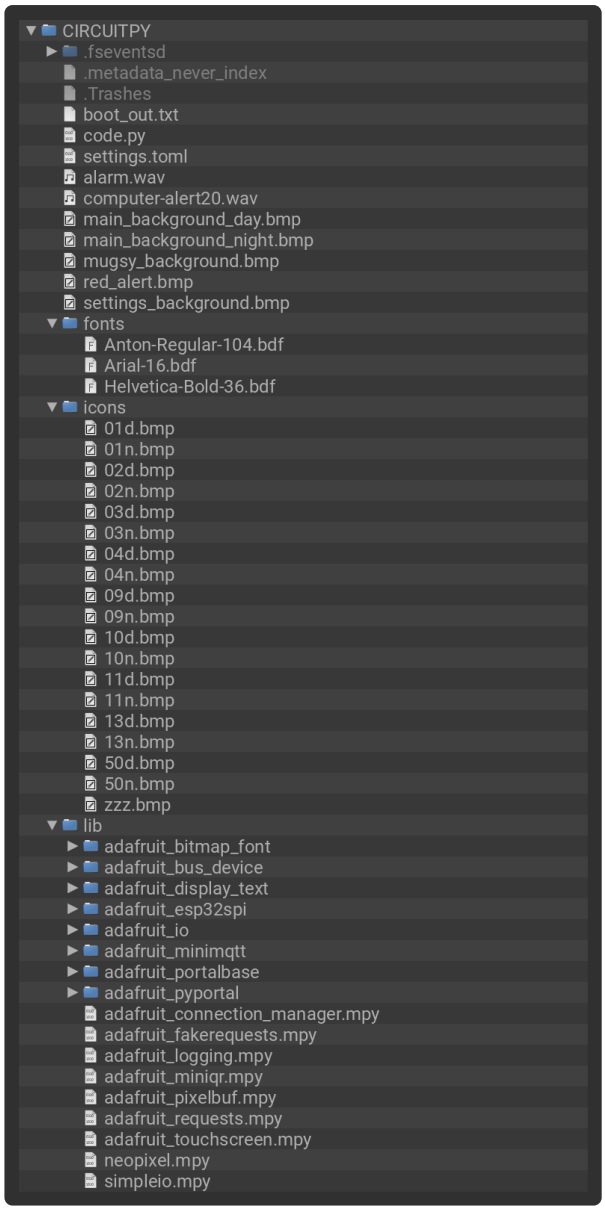
```
            minutes_alarm = alarm_hour * 60 + alarm_minute
            if minutes_now == minutes_alarm:
                if alarm_armed:
                    change_to_state('alarm')
            else:
                alarm_armed = alarm_enabled


    def touch(self, t, touched):
        if t and not touched:             # only process the initial touch
            for button_index in range(len(self.buttons)):
                b = self.buttons[button_index]
                if touch_in_button(t, b):
                    change_to_state(b['next_state'])
                    break
        return bool(t)


    def enter(self):
        self.adjust_backlight_based_on_light(force=True)
        for ta in self.text_areas:
            pyportal.splash.append(ta)
        pyportal.splash.append(self.weather_icon)
        if snooze_time:
            # CircuitPython 6 & 7 compatible
            if self.snooze_file:
                self.snooze_file.close()
            self.snooze_file = open('/icons/zzz.bmp', "rb")
            icon = displayio.OnDiskBitmap(self.snooze_file)
            icon_sprite = displayio.TileGrid(icon,
                                             pixel_shader=getattr(icon,
'pixel_shader', displayio.ColorConverter()))

            # # CircuitPython 7+ compatible
            # icon = displayio.OnDiskBitmap("/icons/zzz.bmp")
            # icon_sprite = displayio.TileGrid(icon, pixel_shader=icon.pixel_shader)

            self.snooze_icon.append(icon_sprite)
            pyportal.splash.append(self.snooze_icon)
        if alarm_enabled:
            self.text_areas[1].text = '%2d:%02d' % (alarm_hour, alarm_minute)
        else:
            self.text_areas[1].text = '     '
        try:
            board.DISPLAY.refresh(target_frames_per_second=60)
        except AttributeError:
            board.DISPLAY.refresh_soon()
            board.DISPLAY.wait_for_frame()



    def exit(self):
        super().exit()
        for _ in range(len(self.snooze_icon)):
            self.snooze_icon.pop()


class Mugsy_State(Time_State):
    """This state tells Mugsey 'Make me a coffee' """

    def __init__(self):
        super().__init__()


    @property
    def name(self):
        return 'mugsy'
```

```
    def tick(self, now):
        # Once the job is done, go back to the main screen
        change_to_state('time')

    def enter(self):
        global low_light
        low_light = False
        pyportal.set_backlight(1.00)
        pyportal.set_background(mugsy_background)
        try:
            board.DISPLAY.refresh(target_frames_per_second=60)
        except AttributeError:
            board.DISPLAY.refresh_soon()
            board.DISPLAY.wait_for_frame()



class Alarm_State(State):
    """This state shows/sounds the alarm.
    Touching anywhere on the screen cancells the alarm.
    Pressing the snooze button turns of the alarm, starting it again in 10
minutes."""

    def __init__(self):
        super().__init__()
        self.sound_alarm_time = None


    @property
    def name(self):
        return 'alarm'


    def tick(self, now):
        global snooze_time

        # is the snooze button pushed
        if not snooze_button.value:
            snooze_time = now
            change_to_state('time')
            return

        # is it time to sound the alarm?
        if self.sound_alarm_time and (now - self.sound_alarm_time) > alarm_interval:
            self.sound_alarm_time = now
            pyportal.play_file(alarm_file)


    def touch(self, t, touched):
        global snooze_time
        if t and not touched:
            snooze_time = None
            change_to_state('time')
        return bool(t)


    def enter(self):
        global low_light
        self.sound_alarm_time = time.monotonic() - alarm_interval
        pyportal.set_backlight(1.00)
        pyportal.set_background(alarm_background)
        low_light = False
        try:
            board.DISPLAY.refresh(target_frames_per_second=60)
        except AttributeError:
            board.DISPLAY.refresh_soon()
            board.DISPLAY.wait_for_frame()
```

```
    def exit(self):
        global alarm_armed
        super().exit()
        alarm_armed = bool(snooze_time)


class Setting_State(State):
    """This state lets the user enable/disable the alarm and set its time.
    Swiping up/down adjusts the hours & miniutes separately."""

    def __init__(self):
        super().__init__()
        self.previous_touch = None
        self.background = 'settings_background.bmp'
        text_area_configs = [dict(x=88, y=120, size=5, color=0xFFFFFF,
font=time_font)]

        self.text_areas = create_text_areas(text_area_configs)
        self.buttons = [dict(left=0, top=30, right=80, bottom=93),    # on
                        dict(left=0, top=98, right=80, bottom=152),   # return
                        dict(left=0, top=155, right=80, bottom=220),  # off
                        dict(left=100, top=0, right=200, bottom = 240), # hours
                        dict(left=220, top=0, right=320, bottom = 240)]   # minutes


    @property
    def name(self):
        return 'settings'


    def touch(self, t, touched):
        global alarm_hour, alarm_minute, alarm_enabled
        if t:
            if touch_in_button(t, self.buttons[0]):   # on
                logger.debug('ON touched')
                alarm_enabled = True
                self.text_areas[0].text = '%02d:%02d' % (alarm_hour, alarm_minute)
            elif touch_in_button(t, self.buttons[1]):   # return
                logger.debug('RETURN touched')
                change_to_state('time')
            elif touch_in_button(t, self.buttons[2]): # off
                logger.debug('OFF touched')
                alarm_enabled = False
                self.text_areas[0].text = '     '
            elif alarm_enabled:
                if not self.previous_touch:
                    self.previous_touch = t
                else:
                    if touch_in_button(t, self.buttons[3]):   # HOURS
                        logger.debug('HOURS touched')
                        if t[1] < (self.previous_touch[1] - 5):   # moving up
                            alarm_hour = (alarm_hour + 1) % 24
                            logger.debug('Alarm hour now: %d', alarm_hour)
                        elif t[1] > (self.previous_touch[1] + 5): # moving down
                            alarm_hour = (alarm_hour - 1) % 24
                            logger.debug('Alarm hour now: %d', alarm_hour)
                        self.text_areas[0].text = '%02d:%02d' % (alarm_hour,
alarm_minute)
                    elif touch_in_button(t, self.buttons[4]): # MINUTES
                        logger.debug('MINUTES touched')
                        if t[1] < (self.previous_touch[1] - 5):   # moving up
                            alarm_minute = (alarm_minute + 1) % 60
                            logger.debug('Alarm minute now: %d', alarm_minute)
                        elif t[1] > (self.previous_touch[1] + 5): # moving down
                            alarm_minute = (alarm_minute - 1) % 60
                            logger.debug('Alarm minute now: %d', alarm_minute)
                        self.text_areas[0].text = '%02d:%02d' % (alarm_hour,
```

```
alarm_minute)
                    self.previous_touch = t
            try:
                board.DISPLAY.refresh(target_frames_per_second=60)
            except AttributeError:
                board.DISPLAY.refresh_soon()
                board.DISPLAY.wait_for_frame()

        else:
            self.previous_touch = None
        return bool(t)


    def enter(self):
        global snooze_time
        snooze_time = None

        pyportal.set_background(self.background)
        for ta in self.text_areas:
            pyportal.splash.append(ta)
        if alarm_enabled:
            self.text_areas[0].text = '%02d:%02d' % (alarm_hour, alarm_minute) #
set time textarea
        else:
            self.text_areas[0].text = '     '


####################
# State management

states = {'time': Time_State(),
          'mugsy': Mugsy_State(),
          'alarm': Alarm_State(),
          'settings': Setting_State()}

current_state = None


def change_to_state(state_name):
    global current_state
    if current_state:
        logger.debug('Exiting %s', current_state.name)
        current_state.exit()
    current_state = states[state_name]
    logger.debug('Entering %s', current_state.name)
    current_state.enter()

####################
# And... go

clear_splash()
change_to_state("time")

while True:
    touched = current_state.touch(pyportal.touchscreen.touch_point, touched)
    current_state.tick(time.monotonic())
```

Your **CIRCUITPY** drive should look something like this after you've copied everything over:



Setup

Imports

We start by importing the library modules we need:

```
import time
import json
from secrets import secrets
import board
from adafruit_pyportal import PyPortal
from adafruit_bitmap_font import bitmap_font
from adafruit_display_text.label import Label
from digitalio import DigitalInOut, Direction, Pull
import analogio
import displayio
import adafruit_logging as logging
```

Notably, the **secrets.py** file contains a dictionary containing hidden information (like api keys, wifi network credentials, etc.) and some general configuration information like location strings for the time and weather services. It's not in the repo, so copy the template below into a text editor and replace the placeholders with your information.

```
secrets = {
    'ssid' : 'your wifi ssid',
    'password' : 'your wifi password',
    'timezone' : None,
    'openweather_token': 'get it from openweather.org',
    'celcius': True,
    'timezone': 'your timezone string',
    'city_id': 'your openweather city id'
}
```

You can find a list of cities and their IDs in http://bulk.openweathermap.org/sample/city.list.json.gz (https://adafru.it/EcE)

Save the file to the main (root) directory of your PyPortal **CIRCUITPY** flash drive.

Configuration and Variables

The data source and location for the PyPortal data fetch support needs to be set up to get the weather information:

```
# Set up where we'll be fetching data from
DATA_SOURCE = 'http://api.openweathermap.org/data/2.5/weather?
id='+secrets['city_id']
DATA_SOURCE += '&amp;appid='+secrets['openweather_token']
# You'll need to get a token from openweather.org, looks like
'b6907d289e10d714a6e88b30761fae22'
DATA_LOCATION = []
```

With that set up, we can create the PyPortal instance as well as the light sensor and snooze button input:

```
pyportal = PyPortal(url=DATA_SOURCE,
                    json_path=DATA_LOCATION,
                    status_neopixel=board.NEOPIXEL)

light = analogio.AnalogIn(board.LIGHT)

snooze_button = DigitalInOut(board.D3)
snooze_button.direction = Direction.INPUT
snooze_button.pull = Pull.UP
```

There is a variety of global variables having to do with managing the alarm, including snoozing:

```
alarm_background = 'red_alert.bmp'
alarm_file = 'alarm.wav'
```

```
alarm_enabled = True
alarm_armed = True
alarm_interval = 10.0
alarm_hour = 7
alarm_minute = 00
snooze_time = None
snooze_interval = 600.0
```

Finally there are variables relating to weather display, update timing, and tracking the light level (to avoid repeatedly updating the display).

```
icon_file = None
icon_sprite = None
celcius = secrets['celcius']

refresh_time = None
update_time = None
weather_refresh = None
current_time = None

low_light = False
```

To provide some visual variation as well as fit more information on the screen, three fonts are used in this project:

1. a large font for displaying the current time,
2. a medium font for displaying the alarm time, and
3. a small font for displaying the temperature.

The time is the main piece of information so we want a big, bold, highly visible font for that. Having the alarm time visible is nice to have, and it also provides a clear indication of whether the alarm is turned on. The weather and temperature aren't crucial, so a smaller font suffices, as well as being sized in keeping with the weather graphic.

```
####################
# Load the fonts

time_font = bitmap_font.load_font('/fonts/Anton-Regular-104.bdf')
time_font.load_glyphs(b'0123456789:') # pre-load glyphs for fast printing

alarm_font = bitmap_font.load_font('/fonts/Helvetica-Bold-36.bdf')
alarm_font.load_glyphs(b'0123456789:')

temperature_font = bitmap_font.load_font('/fonts/Arial-16.bdf')
temperature_font.load_glyphs(b'-0123456789CF')
```

Logging

The logger module (see this guide for information on it (https://adafru.it/Ehw)) is used for debug and error output. It's initial included in the repository, but will be in the

bundle eventually. It needs to be initialized and the logging level set. If you want to see debug information, set this to `logging.DEBUG` instead.

```
logger = logging.getLogger('alarm_clock')
logger.setLevel(logging.ERROR)            # change as desired
```

Support Functions

In addition to the state classes, there are a handful of support functions.

`create_text_areas` created a list of `TextArea` instances based on a list of dictionaries that define the position, font, color, and text length of each area:

```
def create_text_areas(configs):
    """Given a list of area specifications, create and return test areas."""
    text_areas = []
    for cfg in configs:
        textarea = Label(cfg['font'], text=' '*cfg['size'])
        textarea.x = cfg['x']
        textarea.y = cfg['y']
        textarea.color = cfg['color']
        text_areas.append(textarea)
    return text_areas
```

We have a `clear_splash` function that removes display elements from the PyPortal, leaving only the background. Since the background is set up first by the PyPortal code, this can simply pop off elements that have been added since then.

```
def clear_splash():
    for _ in range(len(pyportal.splash) - 1):
        pyportal.splash.pop()
```

Finally we have a function that takes a touch (which is a tuple of x, y, and pressure values) and a button definition (a dictionary with left, right, top, and bottom keys) and returns whether the touch is in the area covered by the button.

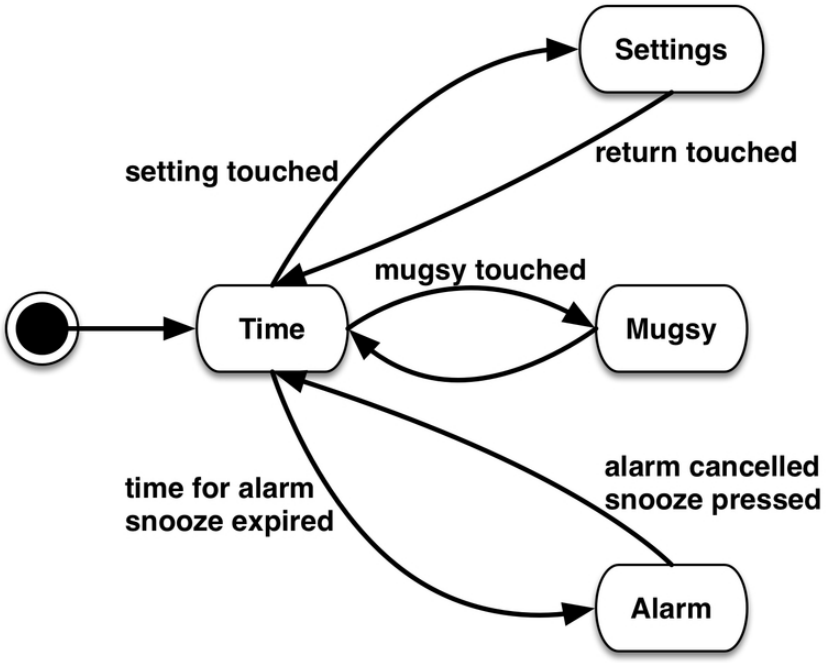
```
def touch_in_button(t, b):
    in_horizontal = b['left'] &lt;= t[0] &lt;= b['right']
    in_vertical = b['top'] &lt;= t[1] &lt;= b['bottom']
    return in_horizontal and in_vertical
```

State Machines

This project is built around a state machine. These were described in detail in a previous guide (https://adafru.it/DtL). We'll use a class based state machine implementation in this project.

This machine has 4 states:

1. The **Time** state is where the program spends most of its time. This state handles the primary screen: displaying the current time and weather, checking whether the alarm should sound, and allowing access to the alarm settings and mugsy (or other) function.
2. The **Mugsy** state is for some additional function. In the author's case it will be telling his Mugsy robotic pour-over machine to start brewing the first mug of the morning. Details of this state are TBD until Mugsy arrives and is up & running.
3. The **Alarm** state handles sounding the alarm, cancelling it by touching the screen, and triggering snooze using the big button.
4. The **Settings** state handles the mode that allows the user to enable or disable the alarm (by touching the red or green areas) and set the alarm time (by swiping up/down over the hourrs and minutes). Touching the yellow arrow "button" returns to the time screen.



The common `State` base class defines the supported methods with their default implementations.

```
class State(object):
    """State abstract base class"""

    def __init__(self):
        pass


    @property
    def name(self):
        """Return the name of teh state"""
        return ''
```

```
    def tick(self, now):
        """Handle a tick: one pass through the main loop"""
        pass


    def touch(self, t, touched):
        """Handle a touch event.
        :param (x, y, z) - t: the touch location/strength"""
        return bool(t)


    def enter(self):
        """Just after the state is entered."""
        pass


    def exit(self):
        """Just before the state exits."""
        clear_splash()
```

__init__() - the constructor. This should create any views, or other objects that live for the life of the system, persisting between times the state is active.

name() - returns the name of the state. This has only used in generating debugging output.

tick(now) - Any time based actions should happen in this method. now is the current value of time.monotonic.

touch(t, touched) - Handle a potential touch. t is the active touch or None. touched is whether there was an active touch last time. The result of this method becomes the touched argument for the next call. This is typically done by returning `bool(t)`, but may not always.

enter() - The state is being entered. If the state needs to display anything, the views need to be added to `pyportal.splash` here.

exit() - The state is being existed. The default here is to remove everything except the background from `pyportal.splash`.

Most of these methods are defaulted to doing nothing. `exit` is the exception in that it cleans up `pyportal.splash`. Because of this, if a state's `exit` method in implemented (because it has something to do), it needs to call `super().exit()`.

The Machine

Management of the states is very lightweight. The states are, themselves, responsible for transitioning. All that is required outside the states is to set up the machine:

```
states = {'time': Time_State(),
          'mugsy': Mugsy_State(),
          'alarm': Alarm_State(),
          'settings': Setting_State()}

current_state = None
```

and provide a way to change states (which takes care of calling exit and enter methods):

```
def change_to_state(state_name):
    global current_state
    if current_state:
        logger.debug('Exiting %s', current_state.name)
        current_state.exit()
    current_state = states[state_name]
    logger.debug('Entering %s', current_state.name)
    current_state.enter()
```

With the machine in place, it's just a matter to starting it in the **time** state and having the core loop simply call `touch` and `tick` on the current state:

```
clear_splash()
change_to_state("time")

while True:
    touched = current_state.touch(pyportal.touchscreen.touch_point, touched)
    current_state.tick(time.monotonic())
```

Details

The next 4 sections will go over the details of each of the states.

The Time State

The Time state is the most complex. It:

- displays the current time, syncing it occasionally from a time service,
- displays the current weather condition and temperature fetched occasionally from a weather service,
- triggers the alarm when the alarm time is reached,
- triggers the alarm when snooze timeout expires, and
- provides access to the settings and mugsy states.

It also has the busiest screen.

`__init__`

The constructor sets up the background files, timers, and screen elements. Notice that each button includes the left, right, top, and bottom of the area it covers, as well as the state to transition to if that button is touched.

```
    def __init__(self):
        super().__init__()
        self.background_day = 'main_background_day.bmp'
        self.background_night = 'main_background_night.bmp'
        self.refresh_time = None
        self.update_time = None
        self.weather_refresh = None
        text_area_configs = [dict(x=88, y=170, size=5, color=0xFFFFFF,
font=time_font),
                             dict(x=210, y=50, size=5, color=0xFF0000,
font=alarm_font),
                             dict(x=88, y=90, size=6, color=0xFFFFFF,
font=temperature_font)]
        self.text_areas = create_text_areas(text_area_configs)
        self.weather_icon = displayio.Group()
        self.weather_icon.x = 88
        self.weather_icon.y = 20
        self.icon_file = None

        self.snooze_icon = displayio.Group()
        self.snooze_icon.x = 260
        self.snooze_icon.y = 70
        self.snooze_file = None

        # each button has it's edges as well as the state to transition to when
touched
        self.buttons = [dict(left=0, top=50, right=80, bottom=120,
next_state='settings'),
                        dict(left=0, top=155, right=80, bottom=220,
next_state='mugsy')]
```

The time state has a helper method to adjust the screen for lighting level. At low levels, it will dim the display backlight and switch to a red-based background. At higher light levels, it sets the display to full brightness and uses a blue-based background.

```
    def adjust_backlight_based_on_light(self, force=False):
        """Check light level. Adjust the backlight and background image if it's
dark."""
        global low_light
        if light.value &lt;= 1000 and (force or not low_light):
            pyportal.set_backlight(0.01)
            pyportal.set_background(self.background_night)
            low_light = True
        elif force or (light.value &gt;= 2000 and low_light):
            pyportal.set_backlight(1.00)
            pyportal.set_background(self.background_day)
            low_light = False
```

`tick`

This is a rather complex method and we'll work through it one piece at a time.

This state has two things to do with snoozing:

1. Check if the snooze button is pressed. If so turn off snooze. If snooze was active, remove the snooze indicator from the screen.
2. Check if the snooze timer has timed out. If so transition to the alarm state.

```
# is the snooze button pushed? Cancel the snooze if so.
        if not snooze_button.value:
            if snooze_time:
                self.snooze_icon.pop()
            snooze_time = None
            alarm_armed = False

        # is snooze active and the snooze time has passed? Transition to alram is
so.
        if snooze_time and ((now - snooze_time) &gt;= snooze_interval):
            change_to_state('alarm')
            return
```

Once snoozing is dealt with, the background is adjusted using the method described earlier.

```
        # check light level and adjust background &amp; backlight
        self.adjust_backlight_based_on_light()
```

Every hour the time is synced with a time service. This keeps the time accurate enough without requiring real time clock (RTC) hardware. It only syncs hourly so as to not consume too much time/power doing the WiFi communication as well as keeping use of the service minimal. This is handled by the `PyPortal` class, so the code here is short:

```
# only query the online time once per hour (and on first run)
        if (not self.refresh_time) or ((now - self.refresh_time) &gt; 3600):
            try:
                pyportal.get_local_time(location=secrets['timezone'])
                self.refresh_time = now
```

```
            except RuntimeError as e:
                self.refresh_time = now - 3000   # delay 10 minutes before retrying
                logger.error('Some error occured, retrying! - %s', str(e))
```

Similarly, the weather data is fetched periodically. In this case. every 10 minutes. Even though this uses the `PyPortal` data fetch and extract support, it is a bit more involved: In addition to formatting and displaying the temperature, the appropriate weather icon needs to be selected and displayed.

```
# only query the weather every 10 minutes (and on first run)
        if (not self.weather_refresh) or (now - self.weather_refresh) > 600:
            logger.debug('Fetching weather')
            try:
                value = pyportal.fetch()
                weather = json.loads(value)

                # set the icon/background
                weather_icon_name = weather['weather'][0]['icon']
                try:
                    self.weather_icon.pop()
                except IndexError:
                    pass
                filename = "/icons/"+weather_icon_name+".bmp"

                if filename:
                    # CircuitPython 6 & 7 compatible
                    if self.icon_file:
                        self.icon_file.close()
                    self.icon_file = open(filename, "rb")
                    icon = displayio.OnDiskBitmap(self.icon_file)

                    icon_sprite = displayio.TileGrid(icon,
                                                     pixel_shader=getattr(icon,
'pixel_shader', displayio.ColorConverter()),
                                                     x=0, y=0)

                    # # CircuitPython 7+ compatible
                    # icon = displayio.OnDiskBitmap(filename)
                    # icon_sprite = displayio.TileGrid(icon,
pixel_shader=icon.pixel_shader)

                    self.weather_icon.append(icon_sprite)

                temperature = weather['main']['temp'] - 273.15 # its...in kelvin
                if celcius:
                    temperature_text = '%3d C' % round(temperature)
                else:
                    temperature_text = '%3d F' % round(((temperature * 9 / 5) + 32))
                self.text_areas[2].text = temperature_text
                self.weather_refresh = now
                try:
                    board.DISPLAY.refresh(target_frames_per_second=60)
                except AttributeError:
                    board.DISPLAY.refresh_soon()
                    board.DISPLAY.wait_for_frame()


            except RuntimeError as e:
                self.weather_refresh = now - 540   # delay a minute before retrying
                logger.error("Some error occured, retrying! - %s", str(e))
```

The final part of this method checks to see if it's time to update the displayed time. If so, it does.

If we are not snoozing. we check whether it is time to sound the alarm (remember that snoozing is handled earlier by using a timer). If so, and the alarm is armed we transition to the alarm state. If it's not alarm time. we arm the alarm (if it's enabled) so that it will sound the next day when the time again reaches the set time.

```
        if (not update_time) or ((now - update_time) &gt; 30):
            # Update the time
            update_time = now
            current_time = time.localtime()
            time_string = '%02d:%02d' % (current_time.tm_hour,current_time.tm_min)
            self.text_areas[0].text = time_string
            board.DISPLAY.refresh_soon()
            board.DISPLAY.wait_for_frame()

            # Check if alarm should sound
        if current_time is not None and not snooze_time:
            minutes_now = current_time.tm_hour * 60 + current_time.tm_min
            minutes_alarm = alarm_hour * 60 + alarm_minute
            if minutes_now == minutes_alarm:
                if alarm_armed:
                    change_to_state('alarm')
            else:
                alarm_armed = alarm_enabled
```

`touch`

This is only concerned with the alarm settings and mugsy buttons. For each button, it checks if the touch (if there is one) is in its area. If so the associated state is transitioned to and the loop terminates.

```
    def touch(self, t, touched):
        if t:
            logger.debug('touched: %d, %d', t[0], t[1])
        if t and not touched:             # only process the initial touch
            for button_index in range(len(self.buttons)):
                b = self.buttons[button_index]
                if touch_in_button(t, b):
                    change_to_state(b['next_state'])
                    break
        return bool(t)
```

`enter`

This is another busy method. It starts by updating the background, forcing the update regardless of what it thinks the light level is.

The text areas created in the constructor are added to the display, as is the weather icon and, if appropriate, the snooze indicator.

If the alarm is enabled, the alarm time is displayed.

Finally the display is updated.

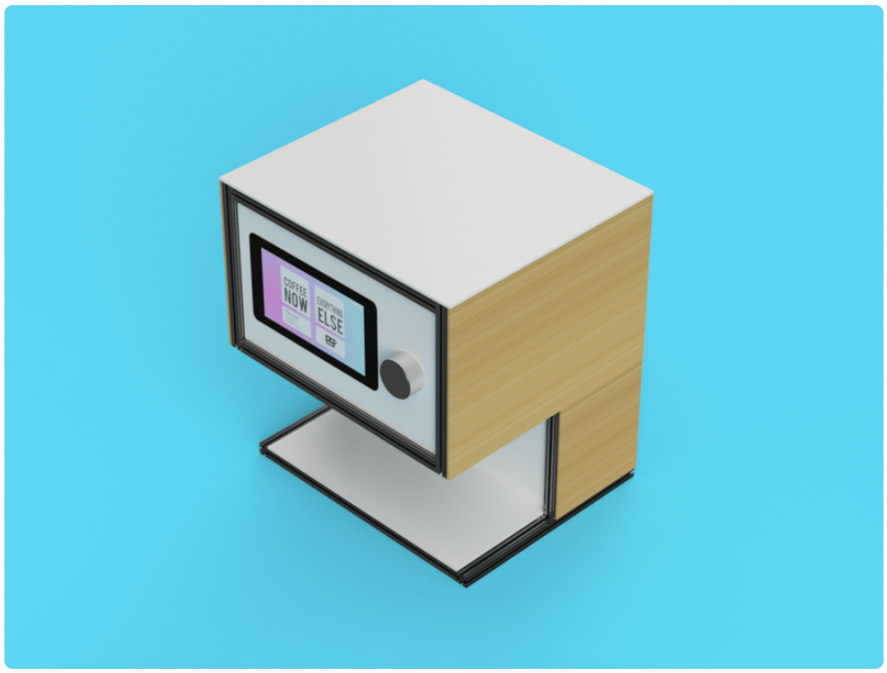
```
def enter(self):
        self.adjust_backlight_based_on_light(force=True)
        for ta in self.text_areas:
            pyportal.splash.append(ta)
        pyportal.splash.append(self.weather_icon)
        if snooze_time:
            # CircuitPython 6 & 7 compatible
            if self.snooze_file:
                self.snooze_file.close()
            self.snooze_file = open('/icons/zzz.bmp', "rb")
            icon = displayio.OnDiskBitmap(self.snooze_file)
            icon_sprite = displayio.TileGrid(icon,
                                             pixel_shader=getattr(icon,
'pixel_shader', displayio.ColorConverter()))

            # # CircuitPython 7+ compatible
            # icon = displayio.OnDiskBitmap("/icons/zzz.bmp")
            # icon_sprite = displayio.TileGrid(icon, pixel_shader=icon.pixel_shader)

            self.snooze_icon.append(icon_sprite)
            pyportal.splash.append(self.snooze_icon)
        if alarm_enabled:
            self.text_areas[1].text = '%2d:%02d' % (alarm_hour, alarm_minute)
        else:
            self.text_areas[1].text = '     '
        try:
            board.DISPLAY.refresh(target_frames_per_second=60)
        except AttributeError:
            board.DISPLAY.refresh_soon()
            board.DISPLAY.wait_for_frame()
```

`exit`

The `exit` method calls the super to clear out the views, as well as clearing out the snooze indicator.

```
    def exit(self):
        super().exit()
        for _ in range(len(self.snooze_icon)):
            self.snooze_icon.pop()
```

The Mugsy State



This state is straightforward, there is no display component, so no screen, and no interaction. All it does is send a command to a Mugsy IoT coffee maker and return to the time state. That's done in `tick`.

The purpose of this state is envisioned to send a "Brew a mug of coffee" command to a Mugsy. However, it can be used for anything similar. For example, it could send a command to IoT lighting or blinds (or both).

```
class Mugsy_State(Time_State):
    """This state tells Mugsey 'Make me a coffee' """

    def __init__(self):
        super().__init__()
        # mugsy support
        self.background = 'mugsy_background.bmp'



    @property
    def name(self):
        return 'mugsy'


    def tick(self, now):
        # Once the job is done, go back to the main screen
        change_to_state('time')

    def enter(self):
        global low_light
        low_light = False
        pyportal.set_backlight(1.00)
        pyportal.set_background(self.background)
```

```
        board.DISPLAY.refresh_soon()
        board.DISPLAY.wait_for_frame()
```

What's this mugsy thing? From the site (https://adafru.it/EcC):

> "**Mugsy** is the world's first hackable, customizable, dead simple, robotic coffee maker. Every aspect of your brew can be controlled, from grind size and water temperature to pour over pattern and bloom time. If you prefer simplicity, Mugsy will get out of the way. Just send it a text, a tweet or hit that **Coffee Now** button and your cup will be ready in the time it takes to get to the kitchen. "

It seems like the ideal thing to have a button on your alarm clock to tell Mugsy to make coffee before you even get out of bed. All that's missing is a robot butler to bring it to you in bed.

The Alarm State

`__init__`

The alarm state starts off simply: the constructor initializes the alarm repeat time (how long between instances of playing the alarm sound file).

```
def __init__(self):
    super().__init__()
    self.sound_alarm_time = None
```

`tick`

The `tick` method does two things.

1. It checks whether the snooze button is pressed. If so, it captures the time and transitions back to the **time** state. Recall that the time state uses `snooze_time` to decide when you reenter the alarm state.
2. It checks to see if it's time to play the alarm sound. If so, it plays it.

```
    def tick(self, now):
        global snooze_time

        # is the snooze button pushed
        if not snooze_button.value:
            snooze_time = now
            change_to_state('time')
            return

        # is it time to sound the alarm?
        if self.sound_alarm_time and (now - self.sound_alarm_time) &gt;
alarm_interval:
            self.sound_alarm_time = now
            pyportal.play_file(alarm_file)
```

`touch`

If there's a touch anywhere on the screen cancel any snooze that might be active and go back the the **time** state.

```
    def touch(self, t, touched):
        global snooze_time
        if t and not touched:
            snooze_time = None
            change_to_state('time')
        return bool(t)
```

`enter`

This sets the alarm time to start the process of playing the alarm sound periodically. It then sets the backlight to full brightness, sets the background to the alarm image, and updates the display.

```
    def enter(self):
        global low_light
        self.sound_alarm_time = time.monotonic()
        pyportal.set_backlight(1.00)
        pyportal.set_background(alarm_background)
        low_light = False
        board.DISPLAY.refresh_soon()
        board.DISPLAY.wait_for_frame()
```

`exit`

Exit calls the base class's implementation and sets whether the alarm is armed based on whether snooze is active.

```
    def exit(self):
        global alarm_armed
        super().exit()
        alarm_armed = bool(snooze_time)
```

The Setting State

`__init__`

The constructor of the setting state sets the background, and sets up to track the changes in consequitive touches. It allocates the single text area (for the alarm time) and sets up buttons:

1. alarm on,
2. return to the time state, and
3. alarm off.

You'll notice that there are two other "buttons" or maybe "touch areas" would be a better term:

1. the hours value, and
2. the minutes value.

These two are the entire height of the display, maximizing the swiping area.

```
    def __init__(self):
        super().__init__()
        self.previous_touch = None
        self.background = 'settings_background.bmp'
        text_area_configs = [dict(x=88, y=120, size=5, color=0xFFFFFF,
font=time_font)]

        self.text_areas = create_text_areas(text_area_configs)
        self.buttons = [dict(left=0, top=30, right=80, bottom=93),    # on
                        dict(left=0, top=98, right=80, bottom=152),   # return
                        dict(left=0, top=155, right=80, bottom=220),  # off
                        dict(left=100, top=0, right=200, bottom = 240), # hours
                        dict(left=220, top=0, right=320, bottom = 240)]   # minutes
```

`touch`

Setting is interesting in that it doesn't have it's own `tick` method. This is because all of its functionality is driven by user interaction. Hence the `touch` method does all the work.

If there is a touch, each button on the left side of the display (on, return, and off, from top to bottom) are check and the appropriate action is taken.

If none of those were touched, and if the alarm is enabled, up and down swipes over the hours or minutes values are checked for. This is where the `previous_touch` variable comes in. It gets updated after checking for the swipe.

The final step in processing a touch is to update the display. If there wasn't a touch (`t` is None), `previous_touch` is reset to `None`.

```
    def touch(self, t, touched):
        global alarm_hour, alarm_minute, alarm_enabled
        if t:
            if touch_in_button(t, self.buttons[0]):   # on
                logger.debug('ON touched')
                alarm_enabled = True
                self.text_areas[0].text = '%02d:%02d' % (alarm_hour, alarm_minute)
            elif touch_in_button(t, self.buttons[1]):   # return
                logger.debug('RETURN touched')
                change_to_state('time')
            elif touch_in_button(t, self.buttons[2]): # off
                logger.debug('OFF touched')
                alarm_enabled = False
                self.text_areas[0].text = '     '
            elif alarm_enabled:
                if not self.previous_touch:
                    self.previous_touch = t
                else:
                    if touch_in_button(t, self.buttons[3]):   # HOURS
                        logger.debug('HOURS touched')
                        if t[1] &lt; (self.previous_touch[1] - 5):   # moving up
                            alarm_hour = (alarm_hour + 1) % 24
                            logger.debug('Alarm hour now: %d', alarm_hour)
                        elif t[1] &gt; (self.previous_touch[1] + 5): # moving down
                            alarm_hour = (alarm_hour - 1) % 24
                            logger.debug('Alarm hour now: %d', alarm_hour)
```

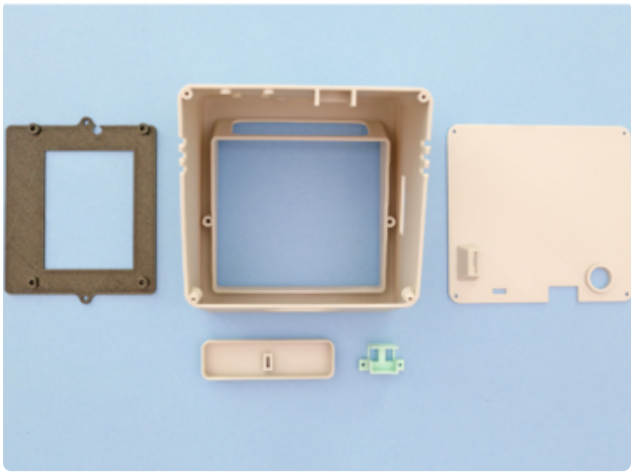
```
                        self.text_areas[0].text = '%02d:%02d' % (alarm_hour,
alarm_minute)
                    elif touch_in_button(t, self.buttons[4]): # MINUTES
                        logger.debug('MINUTES touched')
                        if t[1] &lt; (self.previous_touch[1] - 5):   # moving up
                            alarm_minute = (alarm_minute + 1) % 60
                            logger.debug('Alarm minute now: %d', alarm_minute)
                        elif t[1] &gt; (self.previous_touch[1] + 5): # moving down
                            alarm_minute = (alarm_minute - 1) % 60
                            logger.debug('Alarm minute now: %d', alarm_minute)
                        self.text_areas[0].text = '%02d:%02d' % (alarm_hour,
alarm_minute)
                    self.previous_touch = t
                board.DISPLAY.refresh_soon()
                board.DISPLAY.wait_for_frame()
            else:
                self.previous_touch = None
            return bool(t)
```

`enter`

This sets the background, adds the text area to display the alarm time, and sets it either to spaces or the alarm time depending on if the alarm is enabled.

```
    def enter(self):
        global snooze_time
        snooze_time = None

        pyportal.set_background(self.background)
        for ta in self.text_areas:
            pyportal.splash.append(ta)
        if alarm_enabled:
            self.text_areas[0].text = '%02d:%02d' % (alarm_hour, alarm_minute) #
set time textarea
        else:
            self.text_areas[0].text = '     '
```

3D Printing a Case



3D Printed Parts

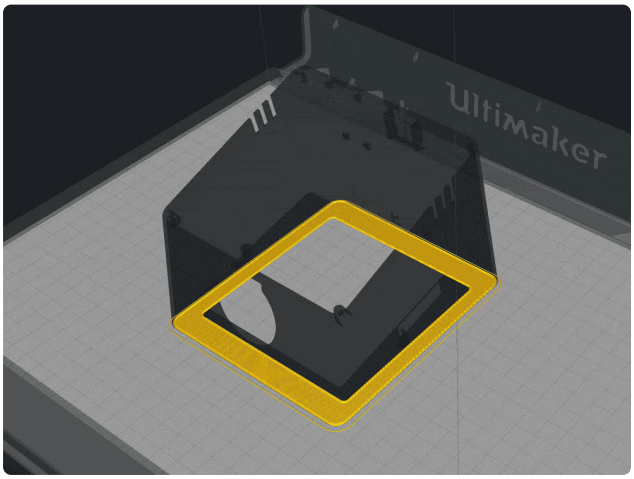
Parts are designed to be 3D printed with FDM based machines. STL files are oriented to print "as is". Parts are listed below with file name and description. Parts require tight tolerances that might need adjusting slice setting. Reference the suggested settings below.

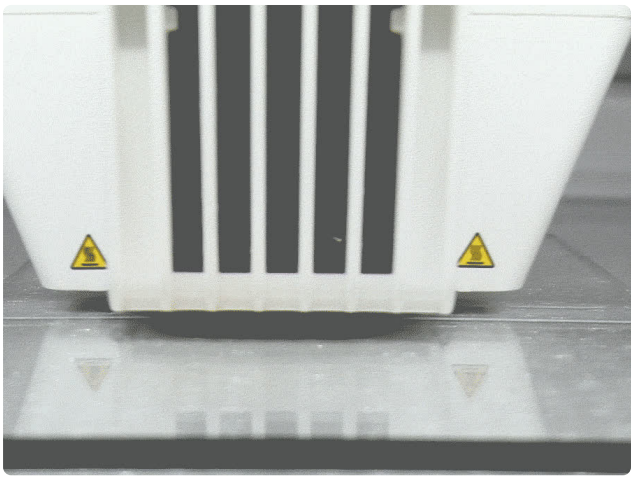
View Design in Fusion360

https://adafru.it/Ekh

Download STLs from Thingiverse

https://adafru.it/EnJ





Slice Settings

Use these settings as reference. Values listed were used in Ultimaker's CURA 3.X slicing software.

0.2mm Layer Height / 0.4mm nozzle
0.35mm Line Width (inner & outer widths)
60mm/s printing speed
20% infill
Supports: Yes

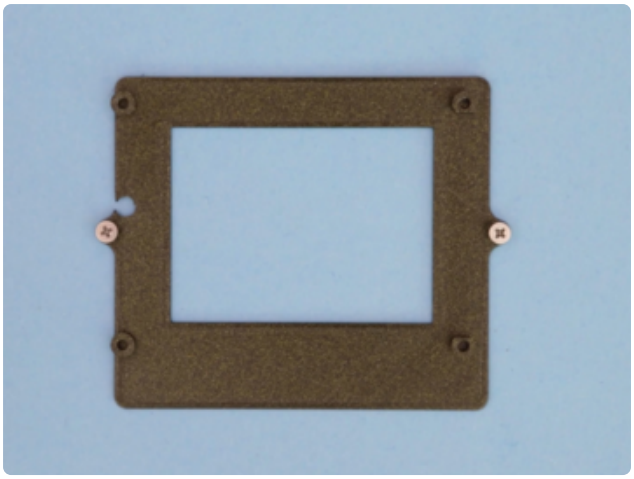
Flat Head Screws

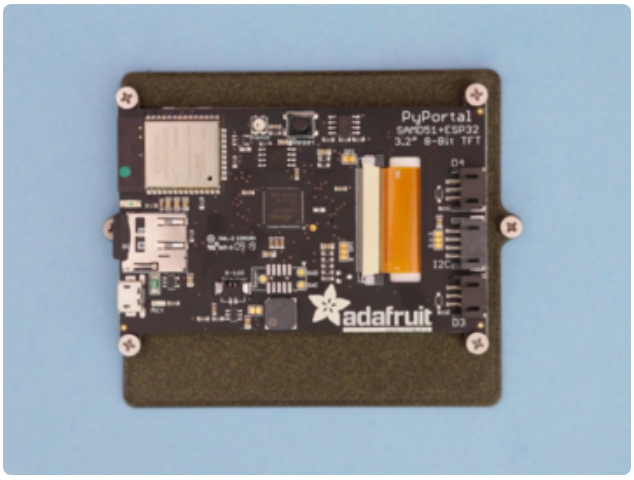
Here are a list of screws used to secure the PyPortal and case parts.

PowerBoost: M2.5x5mm (https://adafru.it/CJl)

PyPortal and bracket: M3x5mm (https://adafru.it/DtD)

Case lid, button: M2x5mm (https://adafru.it/yYe)



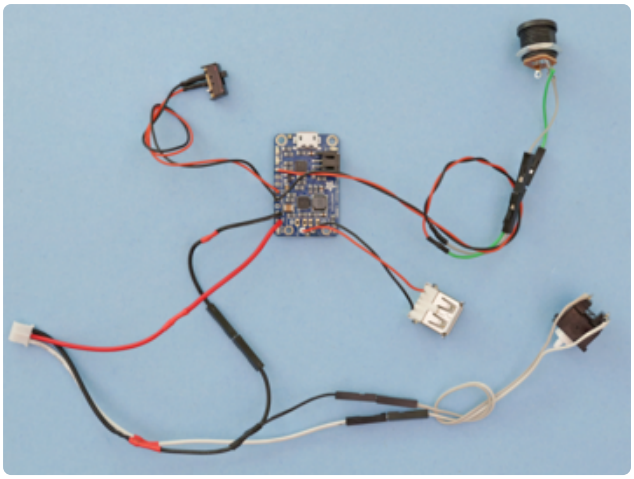


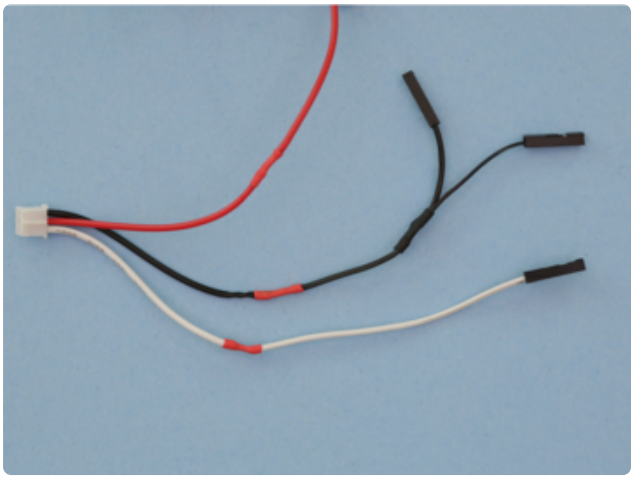
Display Bracket Assembly

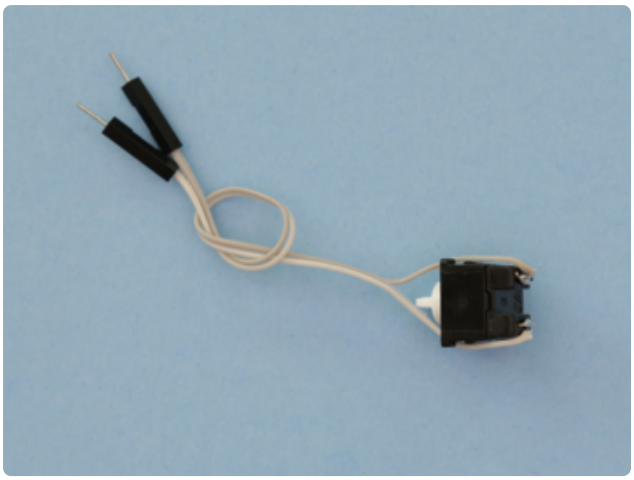
Before mounting our display, we'll need to secure the bracket with two M3x5mm long screws.

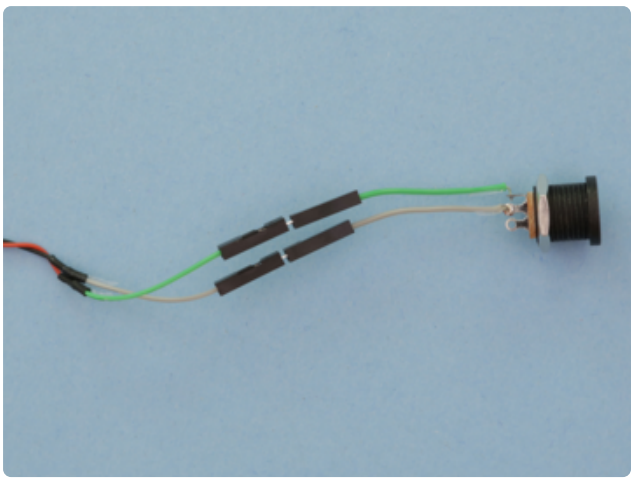
Next, we'll lay the PyPortal on top of the standoffs on the **Bracket** part. The standoffs with the half cuts will align with the display on the PyPortal.

Use four M3x5mm screws to mount the PyPortal to the bracket part.









PowerBoost Connections

To reduce the amount of wires, we trimmed the STEMMA cable short and used female jumper wires to create a "Y" splitter for the ground connection.

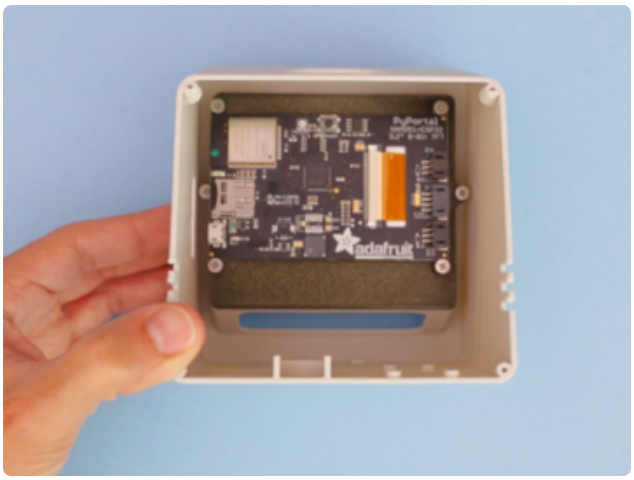
We'll need to add jumper wires to the barrel jack to allow us to mount it through the cases Back lid part.

Here are the wire measurements for each component to solder to the PowerBoost.

USB: 65mm

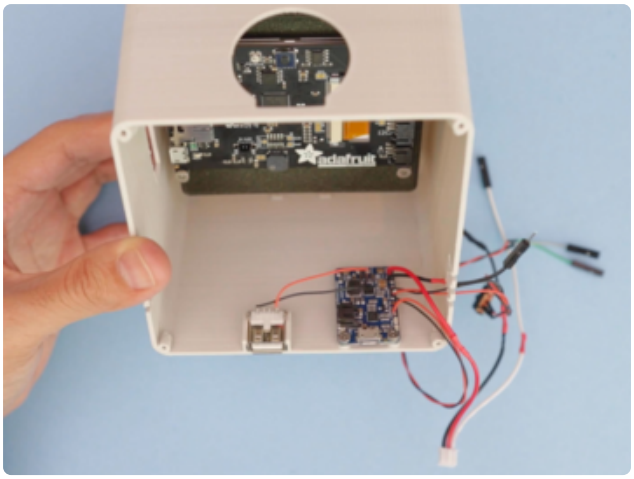
DC Barrel: 180mm

Slide Switch: 180mm



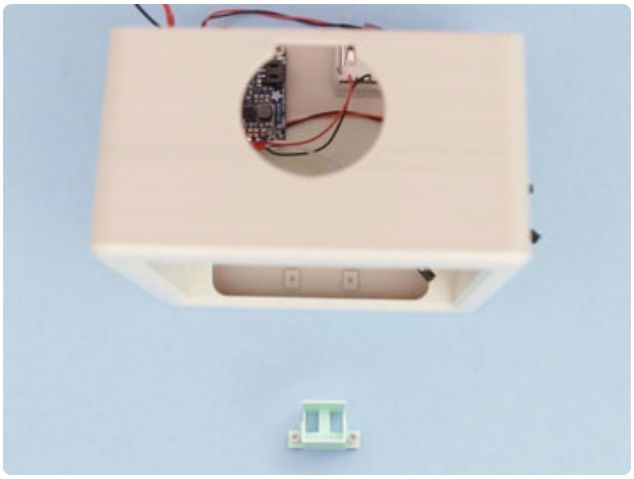
Mount Display

Align the bracket to the two standoffs inside the case. Fasten both screws to secure the display to the case.



Mount PowerBoost

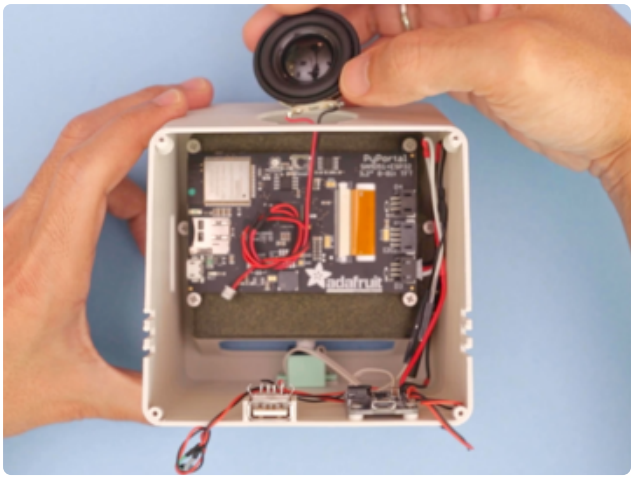
Next, we'll align the USB micro port on the PowerBoost to the edge of the case. Use M2.5x5mm screws to secure to the standoffs. The included USB-A press fits into the three walls on the edge of the case.

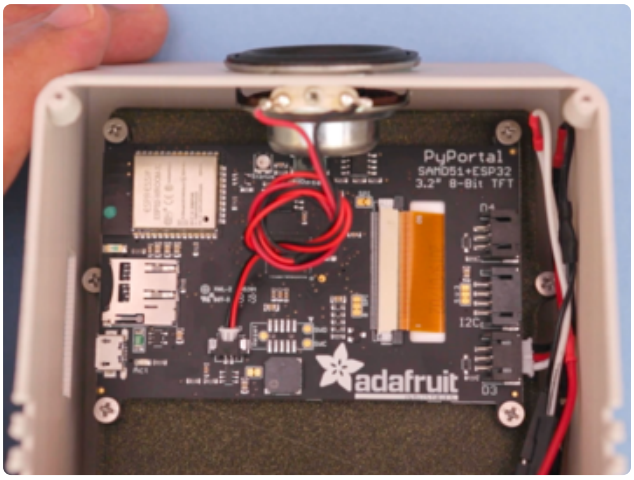


Arcade Button Holder

Our 3d printed key cap replaces the key cap on the arcade button. We'll need to disassemble the arcade button as detailed in the Adafruit NeoPixel Arcade Button Guide (https://adafru.it/EmZ) to remove the button housing to fit the printed key cap.

The **Key Hold** part is secured to the case with two M2x5mm screws.

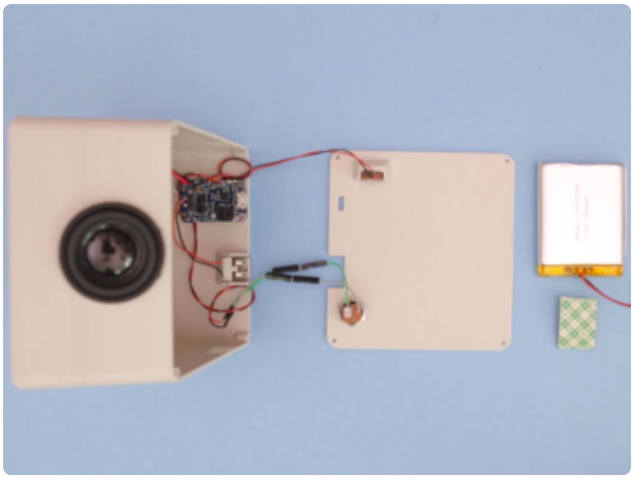


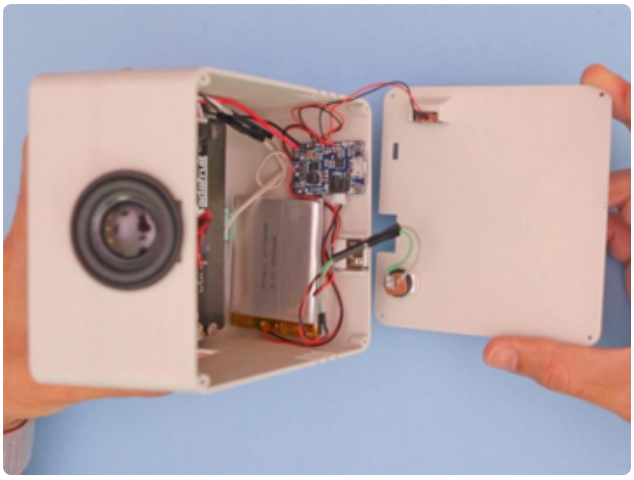


Speaker

After soldering the Pico Blade 2-pin Cable (http://adafru.it/3922) to the speaker, you'll want to coil the wires to keep the wire clutter down.

Align the two tabs on the speaker to the slot on the circular opening on the case. Angle the speaker and press fit into place.





Barrel + Slide Switch

The optional barrel jack mounts to the **Back lid** part. Remove the metal ring around the barrel, pass through the opening and then rethread the metal ring to secure the barrel.

The Slide Switch press fits between the three walls on the Back lid part.

Battery

Attach the battery to the case with a small strip of double stick foam tape.

Mount Key

Finally, align the printed key cap to the mounted actuator. Press fit and test out an alarm!

You can set the alarm by tapping buttons on the screen and use the UI to either turn it on or off.

PyPortal makes building IoT projects fun and CircuitPython makes it easy to learn how to program hardware.

This design has a retro inspired look that fits well with the aesthetic!

An Ongoing Mission...

Where to from here? There are several things you can change to make this project your own:

- the alarm screen
- the alarm sound
- the "Mugsy" function... change it to whatever you want to trigger from the clock. You'll need to edit the backgrounds of the time screen to change the icon.
- actually, anything to do with the screens: images, fonts, and button placement. It's all specified in the constructors of the associated states.