Raspberry Pi E-Ink Event Calendar using Python

Created by Melissa LeBlanc-Williams

https://learn.adafruit.com/raspberry-pi-e-ink-desk-calendar-using-python

Last updated on 2023-08-29 04:29:22 PM EDT
# Table of Contents

## Overview  
- Parts

## Python Setup  
- Wiring  
- Software Setup  
- Python Installation of EPD Library  
- DejaVu TTF Font  
- Pillow Library

## Google Setup
- Project Creation  
- Device Registration  
- Google Calendar API Setup  
- Setup Credentials

## Raspberry Pi Setup
- Install the Google API Package  
- Set your Timezone  
- Upload the Project Files  
- Generating an OAuth Token  
- Resetting your token

## Event Calendar Code
- How It Works  
- Full Example Code
Overview

With this project, you will always know what event you have up next. The eInk Bonnet or Breakout will let you know what's next on your schedule and if you lose power to your Raspberry Pi or other single board computer, you will still be able to see what the next item is because the ePaper display still shows the last thing written to it!

Using Python, this project queries the Google Calendar site API to find out the what is next on your schedule and displays it. This is a great project that you can have sitting on your desk.

The project works by reading the events on your Google Calendar and then sorts what you have coming up and displays the upcoming item as well as the item after that. You can also scroll through events using the buttons.

Parts

To run this, you will need a Single Board Computer such as the Raspberry Pi.
Raspberry Pi 4 Model B - 4 GB RAM
The Raspberry Pi 4 Model B is the newest Raspberry Pi computer made, and the Pi Foundation knows you can always make a good thing better! And what could make the Pi 4 better...
https://www.adafruit.com/product/4296

You will need a 2.13" Monochrome eInk display such as the eInk Bonnet or the eInk Breakout.

Adafruit 2.13" Monochrome eInk / ePaper Display with SRAM
Easy e-paper finally comes to microcontrollers, with this breakout that's designed to make it a breeze to add a monochromatic eInk display. Chances are you've seen one of those...
https://www.adafruit.com/product/4197

If you use a breakout board, you will need a few additional parts. First you will need some tactile switches for buttons:

Tactile Button switch (6mm) x 20 pack
Little clicky switches are standard input "buttons" on electronic projects. These work best in a PCB but
https://www.adafruit.com/product/367

You will also needs a couple of 100K Resistors:
1 x **Full sized breadboard**
Breadboard for assembling parts

1 x **Premium Male/Male Jumper Wires - 40 x 6” (150mm)**
Handy for making wire harnesses or jumpering between headers on PCB's.

1 x **Stacking Header for Pi A+/B+/Pi 2/Pi 3 - 2x20 Extra Tall Header Header**
Header for adding bonnets without interfering with Pi cooling

---

**Python Setup**

**Wiring**

It's easy to use eInk breakouts and bonnets with Python and the Adafruit CircuitPython EPD module. This module allows you to easily write Python code to control the display.

**Using the eInk Bonnet**

Since the eInk Bonnet comes preassembled, all you need to do is place it onto the GPIO pins.

Since there's dozens of Linux computers/boards you can use we will show wiring for Raspberry Pi. For other platforms, please visit the guide for CircuitPython on Linux to see whether your platform is supported.
Connect the display as shown below to your Raspberry Pi.

Using an eInk Breakout

Alternatively, if you have a 2.13" Monochrome eInk Display available, you can wire it up along with a couple of buttons and resistors. There's a lot of wires, which is why we recommend using the bonnet.
3V Rail connects to the Pi's 3V pin
eInk VIN connects to the 3V Rail
eInk GND connects to the Pi's ground
eInk CLK connects to SPI clock. On the Pi, thats SLCK
eInk MOSI connects to SPI MOSI. On the Pi, thats also MOSI
eInk ECS connects to our SPI Chip Select pin. We'll be using CE0
eInk D/C connects to our SPI Chip Select pin. We'll be using GPIO 22.
eInk RST connects to our Reset pin. We'll be using GPIO 13.
GPIO 5 connects to one side of a tactile switch.
GPIO 6 connects to one side of the other tactile switch.
Place a 100KΩ resistor between GPIO5 and the +3V rail.
Place a 100KΩ resistor between GPIO6 and the +3V rail.
Connect the other side of each tactile switch to the Pi's ground.

Note this is not a kernel driver that will let you have the console appear on the TFT. However, this is handy when you can't install an fbtft driver, and want to use the TFT purely from 'user Python' code!

You can only use this technique with Linux/computer devices that have hardware SPI support, and not all single board computers have an SPI device, so check before continuing

Software Setup

You'll need to install the Adafruit_Blinka library that provides the CircuitPython support in Python. This may also require enabling SPI on your platform and verifying you are running Python 3. Since each platform is a little different, and Linux changes often, please visit the CircuitPython on Linux guide to get your computer ready (ﾉ!)
Python Installation of EPD Library

Once that's done, from your command line run the following command:

- `sudo pip3 install adafruit-circuitpython-epd`

If your default Python is version 3 you may need to run `pip` instead. Just make sure you aren't trying to use CircuitPython on Python 2.x, it isn't supported!

If that complains about pip3 not being installed, then run this first to install it:

- `sudo apt-get install python3-pip`

DejaVu TTF Font

Raspberry Pi usually comes with the DejaVu font already installed, but in case it didn't, you can run the following to install it:

- `sudo apt-get install fonts-dejavu`

This package was previously calls ttf-dejavu, so if you are running an older version of Raspberry Pi OS, it may be called that.

Pillow Library

We also need PIL, the Python Imaging Library, to allow graphics and using text with custom fonts. There are several system libraries that PIL relies on, so installing via a package manager is the easiest way to bring in everything:

- `sudo apt-get install python3-pil`

That's it. You should be ready to go!
Google Setup

In order to communicate with the Google Calendar API, you will need to create a project, enable the Google Calendar API, configure all of the permissions and download your OAuth credentials. Since this guide was originally written, Google has changed their API multiple times. You can look at https://developers.google.com/calendar/quickstart/python as an overall reference, but we'll cover the steps below.

Project Creation

Start by going to https://console.actions.google.com/ and clicking New Project.

At the bottom of the page, click on the link to go to Device Registration.
Device Registration

If you have accidentally closed the window, you can get to Device Registration by going back to the Actions Console and selecting your project. After that, click on the Develop tab, the 3-Bar Menu in the upper left, and the Device registration menu item.

At the device registration, click the Register Model button.

Take note of your Model ID in this next step. You will need this in a later step.

Fill in the fields with the requested information. For device type, choose Phone. Make note of your Model ID and save it in a safe place. Click the Register Model Button.
Click Download OAuth 2.0 Credentials and save the JSON file and put it in a safe place such as your Desktop. Save the file as credentials.json. You will upload this to your Raspberry Pi in a later step. Click Next.

Under Traits, click All 7 traits and then click Save Traits

Google Calendar API Setup

Go to the Google Developers Console to enable the API at https://console.developers.google.com/apis/api/embeddedassistant.googleapis.com/overview

Before clicking the button, we need to select the project we created in the first step. By default, the incorrect project is selected.
Next, head over to https://console.cloud.google.com/ () and verify your project is selected. If not, click the project selector at the top.

This opens a dialog box of projects. If the project you created in the first step is not listed here, click inside the search box at the top.

Start typing the name of the project in the search box and click on the result that comes up. Click Open.
Select the 3-Line Menu and then under APIs & Services click on Library.

This will take you to the API Library. In the search box, type Calendar and click on the Google Calendar API result.
Click the Enable button.

Setup Credentials

Once it is enabled, you will be taken to the Overview Screen. Click on the Credentials tab on the left and then click the Configure Consent Screen button on the right.
For User Type, select the option you would like and click Create.

Internal is the preferable option for this type of project because it doesn't require you to submit the app for verification, but this only works for Google Workspace users and not regular Gmail accounts.

Enter an Application Name. This is the name that will appear on the permissions screen.

Select a Support Email. This is the email that will appear when you click on the Application name on the permissions screen.

You may need to add another Email Address under Developer contact information near the bottom of the form.

Scroll to the bottom and Click Save and Continue

Raspberry Pi Setup

First you'll need to install the Adafruit_Blinka library that provides the CircuitPython support in Python. This may also require enabling SPI on your platform and verifying you are running Python 3. Since Linux changes often, please visit the CircuitPython on Linux guide to get your computer ready!
Next you will want to upload the credentials.json file that you saved in the last steps to your Pi into the main folder that you will be running your script. A good place is inside of /home/pi.

**Install the Google API Package**

There is one last set of packages that you will need to install in order to run this project:

```bash
pip3 install --upgrade google-api-python-client google-auth-httplib2 google-auth-oauthlib
```

**Set your Timezone**

If you haven't done so already, be sure your timezone is set correctly. A freshly setup Raspberry Pi is usually set to GMC by default. You can change it by typing:

```bash
sudo raspi-config
```

Then select Localisation Options and Change Time Zone. This will ensure that everything displays with the correct time and date. You can find more information about using raspi-config in the [official documentation](#).

**Upload the Project Files**

In order to communicate with the Google Calendar API, you will need what is called an OAuth token. The first time you run the script (available on the Event Calendar Code page of this guide), you will be given a URL that you can paste into your browser to finish the token setup procedure. Go ahead and upload the script to the same folder that you uploaded the credentials file. Run the script by typing the following:

```bash
python3 event_calendar.py
```

**Generating an OAuth Token**

If you haven't previously run the script, you will be taken through a process that generates an OAuth token, which allows the application to communicate with the Google Calendar API securely.
The script should provide a URL to visit to generate the token. Go ahead and copy and paste the URL into a browser.

You may run into an alert that says your app isn't verified. Go ahead and click on the Advanced link.

That will expand the dialog. Click Go to Quickstart (unsafe) or whatever you decided on for a project name.
Next, you will be asked to grant permissions to view your calendar. Go ahead and click Allow.

After all that, it will come up with a confirmation dialog with your account name and the permissions that you are granting. Go ahead and click on Allow.

Finally, you will be given an Authorization Code. Click on the Copy icon and it will get copied to your clipboard.
Paste the Authorization Code back into the Python script and the token will be generated and saved for next time.

The script will automatically resume and your E-Ink display should be updated with the next event in your calendar.

You should now see something similar to this on your ePaper display.

Resetting your token

If for whatever reason you would like reset your token, you can do so by removing the file token.pickle from the same folder where you ran the script:

```
rm token.pickle
```

Just re-run the event calendar to go through the steps again.

---

Event Calendar Code

Be sure to go through the Google Setup to generate an OAuth token so that this project can read your calendars. Once you have it setup, running the script will not require token generation.

Let's take quick a look at operation of the calendar and then we can take a deeper look at the code that makes it operate.
Upon starting the script, it will check that a token exists. If not, it will go through the token creation process described on the previous page. Once there's a token, it will go out to Google and retrieve all of the calendar IDs and then grab the events from each calendar and sort them. It will start with displaying the item with the next starting time first and you can scroll through items using the buttons.

The script will continue to go out to Google and retrieve any events. This way you can add, modify, or remove events from your calendar and the changes will be reflected on the display.

How It Works

First we start by loading any libraries that are used. This project uses quite a few libraries including the datetime library for date and time interval calculations, the pickle library for reading and writing the token data, several google api libraries, the textwrap library for wrapping text nicely, PIL which is the Python Image Library for laying out the text nicely, and the adafruit_epd library for writing to the ePaper Display.

```python
from __future__ import print_function
from datetime import datetime
import time
import pickle
import os.path
from googleapiclient.discovery import build
from google_auth_oauthlib.flow import InstalledAppFlow
from google.auth.transport.requests import Request
import textwrap
import digitalio
import busio
import board
from PIL import Image, ImageDraw, ImageFont
```
Next, we setup SPI and any pins used. If you are using the EInk bonnet or have wired it up like in the setup page, you shouldn't need to change anything unless you are using a different board than the Raspberry Pi.

```python
spi = busio.SPI(board.SCK, MOSI=board.MOSI, MISO=board.MISO)
ecs = digitalio.DigitalInOut(board.CE0)
dc = digitalio.DigitalInOut(board.D22)
rst = digitalio.DigitalInOut(board.D27)
busy = digitalio.DigitalInOut(board.D17)
up_button = digitalio.DigitalInOut(board.D5)
up_button.switch_to_input()
down_button = digitalio.DigitalInOut(board.D6)
down_button.switch_to_input()
```

Next we define `SCOPES`, which is the permissions we are requesting. It's always a good idea to only use the minimum permissions required.

```python
# If modifying these scopes, delete the file token.pickle.
SCOPES = [
  "https://www.googleapis.com/auth/calendar.readonly"
]
```

The next section contains the parameters that can be changed to alter the script. The `QUERY_DELAY` is the amount of time in seconds that the script will wait before it checks google again. Google has limits to how often you can perform a query, so it's best not to set this too low, especially if you have a lot of calendars.

The `MAX_EVENTS_PER_CAL` is the maximum number of items to retrieve per calendar. If you have a lot of items each day, you may want to increase this.

The `MAX_LINES` is the number of lines of text to display for the next event. Setting this too high may cause text to overlap.

The `DEBOUNCE_DELAY` is the amount of time to wait after a button press is detected. Setting this too low can cause double taps to be falsely detected and setting this too high can make the buttons feel non-responsive.

```python
# Check for new/deleted events every 10 seconds
QUERY_DELAY = 10  # Time in seconds to delay between querying the Google Calendar API
MAX_EVENTS_PER_CAL = 5
MAX_LINES = 2
DEBOUNCE_DELAY = 0.3
```

After that, we setup the ePaper display and set the rotation. If you are using an older eInk Bonnet, you will need to comment out the line for the newer bonnet and uncomment the one for the older bonnet.
# Initialize the Display
```python
display = Adafruit_SSD1680(  # Newer eInk Bonnet
display = Adafruit_SSD1675(  # Older eInk Bonnet
122, 250, spi, cs_pin=ecs, dc_pin=dc, sramcs_pin=None, rst_pin=rst,
busy_pin=busy,
)

display.rotation = 1
```

Next we define a couple of colors to make the code more readable:
```python
# RGB Colors
WHITE = (255, 255, 255)
BLACK = (0, 0, 0)
```

The next section checks if a token is there and runs the setup process if not. Much of this functionality is contained in the Google libraries. This was taken from the Quickstart example and modified to use the `flow.run_console()` function so that it would run easily on a Raspberry Pi console.

```python
creds = None
# The file token.pickle stores the user's access and refresh tokens, and is
# created automatically when the authorization flow completes for the first
time.
if os.path.exists("token.pickle"):
  with open("token.pickle", "rb") as token:
    creds = pickle.load(token)
# If there are no (valid) credentials available, let the user log in.
if not creds or not creds.valid:
  if creds and creds.expired and creds.refresh_token:
    creds.refresh(Request())
  else:
    flow = InstalledAppFlow.from_client_secrets_file("credentials.json", SCOPES)
    creds = flow.run_console()
# Save the credentials for the next run
with open("token.pickle", "wb") as token:
    pickle.dump(creds, token)
```

Next we connect to the Google API service using the credentials and set a few variables with some initial values.

```python
service = build("calendar", "v3", credentials=creds)

current_event_id = None
last_check = None
events = []
```

Now we have our first function `display_event()`. This takes care of drawing out everything to the eInk display for the specified event_id. PIL's `Image.new()` function is used to create a canvas the size of the display and then the `draw.text()` function is used to draw the text to the canvas in various locations. Once all the text is drawn, the canvas is passed into the EPD library with the `self.display.image()` function and then the display is refreshed with the `self.display.display()` function.

©Adafruit Industries
def display_event(event_id):
    event_index = search_id(event_id)
    if event_index is None:
        if len(events) > 0:
            # Event was probably deleted while we were updating
            event_index = 0
            event = events[0]
        else:
            event = None
    else:
        event = events[event_index]

    current_time = get_current_time()
    display.fill(Adafruit_EPD.WHITE)
    image = Image.new("RGB", (display.width, display.height), color=WHITE)
    draw = ImageDraw.Draw(image)
    event_font = ImageFont.truetype("/usr/share/fonts/truetype/dejavu/DejaVuSans.ttf", 24)
    time_font = ImageFont.truetype("/usr/share/fonts/truetype/dejavu/DejaVuSans-Bold.ttf", 18)
    next_event_font = ImageFont.truetype("/usr/share/fonts/truetype/dejavu/DejaVuSans-Bold.ttf", 16)

    # Draw Time
    current_time = get_current_time()
    (font_width, font_height) = time_font.getsize(current_time)
    draw.text((display.width - font_width - 2, 2), current_time, font=time_font, fill=BLACK)

    if event is None:
        text = "No events found"
        (font_width, font_height) = event_font.getsize(text)
        draw.text((display.width // 2 - font_width // 2, display.height // 2 - font_height // 2), text, font=event_font, fill=BLACK)
    else:
        how_long = format_interval(event["start"].get("dateTime", event["start"].get("date"))
        draw.text((2, 2), how_long, font=time_font, fill=BLACK)

        (font_width, font_height) = event_font.getsize(event["summary"])
        lines = textwrap.wrap(event["summary"], width=20)
        for line_index, line in enumerate(lines):
            if line_index < MAX_LINES:
                draw.text((2, line_index * font_height + 22), line, font=event_font, fill=BLACK)

        # Draw Next Event if there is one
        if event_index < len(events) - 1:
            next_event = events[event_index + 1]
Next is the \texttt{format\_event\_date()} function. This function will take a date string and depending on whether it is for today or a future date, it will either display the just the time or the date and time respectively.

```python
def format_event_date(datestr):
    event_date = datetime.fromisoformat(datestr)
    # If the same day, just return time
    if event_date.date() == datetime.now().date():
        return event_date.strftime("%I:%M %p")
    # If a future date, return date and time
    return event_date.strftime("%m/%d/%y %I:%M %p")
```

After that is the \texttt{format\_interval()} function. This function will figure out how far the event is in the future and display whether the event is happening now or in a specified number of minutes, hours, or days.

```python
def format_interval(datestr):
    event_date = datetime.fromisoformat(datestr).replace(tzinfo=None)
    delta = event_date - datetime.now()
    # if < 60 minutes, return minutes
    if delta.days < 0:
        return "Now:"
    if not delta.days and delta.seconds < 3600:
        value = round(delta.seconds / 60)
        return "In {} minute{}:".format(value, "s" if value &gt; 1 else ")"
    # if &lt; 24 hours return hours
    if not delta.days:
        value = round(delta.seconds / 3600)
        return "In {} hour{}:".format(value, "s" if value &gt; 1 else ")"
    return "In {} day{}:".format(delta.days, "s" if delta.days &gt; 1 else ")"
```

The \texttt{search\_id()} function just checks for an \texttt{event\_id} and if it can't find it, it will return None. This usually only happens when the current event is deleted from the calendar.

```python
def search_id(event_id):
    if event_id is not None:
```
for index, event in enumerate(events):
    if event['id'] == event_id:
        return index
return None

The **get_current_time()** function will grab the current time and return it as a formatted string.

```python
def get_current_time():
    now = datetime.now()
    return now.strftime("%I:%M %p")
```

The **get_events()** function will get all the events up to the value of **MAX_EVENTS_PER_CAL** for the given **calendar_id** and return them as a list. This is used to make the main loop smaller and easier to read.

```python
def get_events(calendar_id):
    print("Fetching Events for {}".format(calendar_id))
    page_token = None
    events = (service.events().list(
        calendarId=calendar_id,
        timeMin=now,
        maxResults=MAX_EVENTS_PER_CAL,
        singleEvents=True,
        orderBy="startTime",
    ).execute())
    return events.get("items", [])
```

The **get_all_calendar_ids()** function will get all the Calendar IDs that it can find. This allows the project to read more than a single calendar. This is also used to make the main loop smaller and easier to read.

```python
def get_all_calendar_ids():
    page_token = None
    calendar_ids = []
    while True:
        print("Fetching Calendar IDs")
        calendar_list = service.calendarList().list(pageToken=page_token).execute()
        for calendar_list_entry in calendar_list["items"]: calendar_ids.append(calendar_list_entry["id"]) page_token = calendar_list.get("nextPageToken")
    if not page_token: break
    return calendar_ids
```

Finally, we get to the main loop. We'll break this down a bit since it's a little on the longer side. First it sets what the last event id and last time were so these values can be compared with any new ones.
while True:
    last_event_id = current_event_id
    last_time = current_time

    Next it checks if the amount of time specified in `QUERY_DELAY` delay has passed and if so, it checks the Google API and downloads the data using the `get_all_calendar_ids()` and `get_events()` functions defined above. Once it grabs all the events from the calendars, the entire list is sorted using the built-in `sorted()` function by the start date.

    if last_check is None or time.monotonic() >= last_check + QUERY_DELAY:
        # Call the Calendar API
        now = datetime.utcnow().isoformat() + "Z"
        calendar_ids = get_all_calendar_ids()
        events = []
        for calendar_id in calendar_ids:
            events += get_events(calendar_id)

        # Sort Events by Start Time
        events = sorted(events, key=lambda k: k["start"].get("dateTime", k["start"].get("date"))
        last_check = time.monotonic()

        # Update the current time
        current_time = get_current_time()

    Once the events are updated, the `current_event_id` and `current_index` are updated depending on the results. At this point, the buttons are also checked and debounced using `DEBOUNCE_DELAY`. If a button is pressed, the `current_index` is changed and the appropriate event is displayed assuming we aren't trying to go past the limits of the number of events found.

    if not events:
        current_event_id = None
        current_index = None
    else:
        if current_event_id is None:
            current_index = 0
        else:
            current_index = search_id(current_event_id)

        if current_index is not None:
            # Check for Button Presses
            if up_button.value != down_button.value:
                if not up_button.value and current_index < len(events) - 1:
                    current_index += 1
                    time.sleep(DEBOUNCE_DELAY)
                if not down_button.value and current_index > 0:
                    current_index -= 1
                    time.sleep(DEBOUNCE_DELAY)

            current_event_id = events[current_index]["id"]
        else:
            current_event_id = None
Finally, we check if we actually are supposed to show a different event than the current one displayed and if so, update it using `display_event()`. The main reason we do this check is to avoid unnecessarily updating the eInk display because updating it too often can result in permanent damage to the display.

```python
if current_event_id != last_event_id or current_time != last_time:
    display_event(current_event_id)
```

**Full Example Code**

```python
# SPDX-FileCopyrightText: 2020 Melissa LeBlanc-Williams for Adafruit Industries
# # SPDX-License-Identifier: MIT
from __future__ import print_function
from datetime import datetime
import time
import pickle
import os.path
from googleapiclient.discovery import build
from google_auth_oauthlib.flow import InstalledAppFlow
from google.auth.transport.requests import Request
import textwrap
import digitalio
import busio
import board
from PIL import Image, ImageDraw, ImageFont
from adafruit_epd.epd import Adafruit_EPD
from adafruit_epd.ssd1675 import Adafruit_SSD1675
from adafruit_epd.ssd1680 import Adafruit_SSD1680

spi = busio.SPI(board.SCK, MOSI=board.MOSI, MISO=board.MISO)
ecs = digitalio.DigitalInOut(board.CE0)
dc = digitalio.DigitalInOut(board.D22)
rst = digitalio.DigitalInOut(board.D27)
busy = digitalio.DigitalInOut(board.D17)
up_button = digitalio.DigitalInOut(board.D5)
up_button.switch_to_input()
down_button = digitalio.DigitalInOut(board.D6)
down_button.switch_to_input()

# If modifying these scopes, delete the file token.pickle.
SCOPES = ["https://www.googleapis.com/auth/calendar.readonly"]
# Check for new/deleted events every 10 seconds
QUERY_DELAY = 10  # Time in seconds to delay between querying the Google Calendar API
MAX_EVENTS_PER_CAL = 5
MAX_LINES = 2
DEBOUNCE_DELAY = 0.3

# Initialize the Display
display = Adafruit_SSD1680(  # Newer eInk Bonnet
display = Adafruit_SSD1675(  # Older eInk Bonnet
   122, 250, spi, cs_pin=ecs, dc_pin=dc, sramcs_pin=None, rst_pin=rst,
   busy_pin=busy,
)

display.rotation = 1

# RGB Colors
WHITE = (255, 255, 255)
```
creds = None
# The file token.pickle stores the user's access and refresh tokens, and is
created automatically when the authorization flow completes for the first
time.
if os.path.exists("token.pickle"):
    with open("token.pickle", "rb") as token:
        creds = pickle.load(token)
# If there are no (valid) credentials available, let the user log in.
if not creds or not creds.valid:
    if creds and creds.expired and creds.refresh_token:
        creds.refresh(Request())
    else:
        flow = InstalledAppFlow.from_client_secrets_file("credentials.json", SCOPES)
        creds = flow.run_console()
    # Save the credentials for the next run
    with open("token.pickle", "wb") as token:
        pickle.dump(creds, token)

service = build("calendar", "v3", credentials=creds)

current_event_id = None
last_check = None
events = []

def display_event(event_id):
    event_index = search_id(event_id)
    if event_index is None:
        if len(events) > 0:
            # Event was probably deleted while we were updating
            event_index = 0
            event = events[0]
        else:
            event = None
    else:
        event = events[event_index]

    current_time = get_current_time()
    display.fill(Adafruit_EPD.WHITE)
    image = Image.new("RGB", (display.width, display.height), color=WHITE)
    draw = ImageDraw.Draw(image)
    event_font = ImageFont.truetype(
        "/usr/share/fonts/truetype/dejavu/DejaVuSans.ttf", 24
    )
    time_font = ImageFont.truetype(
        "/usr/share/fonts/truetype/dejavu/DejaVuSans-Bold.ttf", 18
    )
    next_event_font = ImageFont.truetype(
        "/usr/share/fonts/truetype/dejavu/DejaVuSans-Bold.ttf", 16
    )

    # Draw Time
    current_time = get_current_time()
    (font_width, font_height) = time_font.getsize(current_time)
    draw.text((display.width - font_width - 2, 2), current_time, font=time_font,
        fill=BLACK,
    )
    if event is None:
        text = "No events found"
        (font_width, font_height) = event_font.getsize(text)
        draw.text(
            
            display.width // 2 - font_width // 2,
            display.height // 2 - font_height // 2,
            ),
        ),
else:
    how_long = format_interval(
        event["start"]["dateTime", event["start"]["date"]]
    )
    draw.text(
        (2, 2), how_long, font=time_font, fill=BLACK,
    )

    (font_width, font_height) = event_font.getsize(event["summary"])
    lines = textwrap.wrap(event["summary"], width=20)
    for line_index, line in enumerate(lines):
        if line_index < MAX_LINES:
            draw.text(
                (2, line_index * font_height + 22),
                line,
                font=event_font,
                fill=BLACK,
            )

    # Draw Next Event if there is one
    if event_index < len(events) - 1:
        next_event = events[event_index + 1]
        next_time = format_event_date(
            next_event["start"]["dateTime", next_event["start"]["date"]]
        )
        next_item = "Then " + next_time + ": "
        (font_width, font_height) = next_event_font.getsize(next_item)
        draw.text(
            (2, display.height - font_height * 2 - 8),
            next_item,
            font=next_event_font,
            fill=BLACK,
        )
        draw.text(
            (2, display.height - font_height - 2),
            next_event["summary"],
            font=next_event_font,
            fill=BLACK,
        )

display.image(image)
display.display()
def search_id(event_id):
    if event_id is not None:
        for index, event in enumerate(events):
            if event["id"] == event_id:
                return index
        return None

def get_current_time():
    now = datetime.now()
    return now.strftime("%I:%M %p")

current_time = get_current_time()

def get_events(calendar_id):
    print("Fetching Events for {}".format(calendar_id))
    page_token = None
    events = (
        service.events()
            .list(
                calendarId=calendar_id,
                timeMin=now,
                maxResults=MAX_EVENTS_PER_CAL,
                singleEvents=True,
                orderBy="startTime",
            ).execute()
    )
    return events.get("items", [])

def get_all_calendar_ids():
    page_token = None
    calendar_ids = []
    while True:
        print("Fetching Calendar IDs")
        calendar_list = service.calendarList().list(pageToken=page_token).execute()
        for calendar_list_entry in calendar_list["items"]:  # Add calendar_list_entry["id"]
            calendar_ids.append(calendar_list_entry["id"])  # Add calendar_list_entry["id"]
        page_token = calendar_list.get("nextPageToken")
        if not page_token:
            break
    return calendar_ids

while True:
    last_event_id = current_event_id
    last_time = current_time
    if last_check is None or time.monotonic() >= last_check + QUERY_DELAY:
        # Call the Calendar API
        now = datetime.utcnow().isoformat() + "Z"
        calendar_ids = get_all_calendar_ids()
        events = []
        for calendar_id in calendar_ids:
            events += get_events(calendar_id)
        # Sort Events by Start Time
        events = sorted(events, key=lambda k: k["start"]["dateTime", k["start"]["date"])
    last_check = time.monotonic()
# Update the current time
current_time = get_current_time()

if not events:
    current_event_id = None
    current_index = None
else:
    if current_event_id is None:
        current_index = 0
    else:
        current_index = search_id(current_event_id)

if current_index is not None:
    # Check for Button Presses
    if up_button.value != down_button.value:
        if not up_button.value and current_index < len(events) - 1:
            current_index += 1
            time.sleep(DEBOUNCE_DELAY)
        if not down_button.value and current_index > 0:
            current_index -= 1
            time.sleep(DEBOUNCE_DELAY)

    current_event_id = events[current_index]["id"]
else:
    current_event_id = None

if current_event_id != last_event_id or current_time != last_time:
display_event(current_event_id)