Adafruit GPS Library Documentation

Release 1.0

Tony DiCola, James Carr

Oct 05, 2021

Contents

GPS parsing module. Can send commands to, and parse simple NMEA data sentences from serial and I2C GPS modules to read latitude, longitude, and more.

CHAPTER 1

Dependencies

This driver depends on:

- Adafruit CircuitPython

Please ensure all dependencies are available on the CircuitPython filesystem. This is easily achieved by downloading the Adafruit library and driver bundle.

CHAPTER 2

Installing from PyPI

On supported GNU/Linux systems like the Raspberry Pi, you can install the driver locally from PyPI. To install for current user:

```
pip3 install adafruit-circuitpython-gps
```

To install system-wide (this may be required in some cases):

```
sudo pip3 install adafruit-circuitpython-gps
```

To install in a virtual environment in your current project:

```
mkdir project-name && cd project-name
python3 -m venv .env
source .env/bin/activate
pip3 install adafruit-circuitpython-gps
```

CHAPTER 3

Usage Example

See examples/gps_simpletest.py for a demonstration of parsing and printing GPS location.

Important: Feather boards and many other circuitpython boards will round to two decimal places like this:

```
>>> float('1234.5678')
1234.57
```

This isn't ideal for GPS data as this lowers the accuracy from 0.1m to 11m.

This can be fixed by using string formatting when the GPS data is output.

An implementation of this can be found in examples/gps_simpletest.py

```
import time
import board
import busio

import adafruit_gps

RX = board.RX
TX = board.TX

uart = busio.UART(TX, RX, baudrate=9600, timeout=30)

gps = adafruit_gps.GPS(uart, debug=False)

gps.send_command(b'PMTK314,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0')

gps.send_command(b'PMTK220,1000')

last_print = time.monotonic()
while True:

    gps.update()
```

(continues on next page)

(continued from previous page)

```
    current = time.monotonic()
    if current - last_print >= 1.0:
        last_print = current
        if not gps.has_fix:
            print('Waiting for fix...')
            continue
        print('=' * 40)  # Print a separator line.
        print('Latitude: {0:.6f} degrees'.format(gps.latitude))
        print('Longitude: {0:.6f} degrees'.format(gps.longitude))
```

These two lines are the lines that actually solve the issue:

```
print('Latitude: {0:.6f} degrees'.format(gps.latitude))
print('Longitude: {0:.6f} degrees'.format(gps.longitude))
```

Note: Sending multiple PMTK314 packets with `gps.send_command()` will not work unless there is a substantial amount of time in-between each time `gps.send_command()` is called. A `time.sleep()` of 1 second or more should fix this.

CHAPTER 4

Contributing

Contributions are welcome! Please read our Code of Conduct before contributing to help this project stay welcoming.

CHAPTER 5

Documentation

For information on building library documentation, please check out this guide.

CHAPTER 6

Table of Contents

6.1 Communicating with the GPS

The code communicates with the GPS by sending and receiving specially formatted sentences. The format used is the NMEA 0183 protocol specified by the National Marine Electronics Association. This was designed for boat navigation and control systems and is widely used by GPSs.

In general, you configure the device to send the sentences that you want at the frequency you need and then receive a flow of GPS update messages.

Sentences received from the GPS module use the same format, irrespective of the manufacturer. Sentences sent to the GPS module to control it, and answers to these commands, are proprietary to the manufacturer.

NOTE: All of the example commands used in this documentation, and the examples folder, are for the MediaTek 333X GPS chips used in Adafruit products. Make sure to check the datasheet for your GPS chip if it is different.

6.1.1 Sentence format

$TAG[,DATA[,DATA...]]*hh<CR><LF>

- '$' is the opening delimiter
- TAG is the tag describing the type of message.
  - The tag for a proprietary (chipset specific) message is composed of
    - 'P' for proprietary.
    - 'ABC', a 3 letter code for the manufacturer, eg. 'MTK' for MediaTek.
    - 'CODE', a manufacturer specified code for the command or answer. *Note: This can be made up of letters and numbers and there is no required length.*

    'PMTK220' is the Mediatek command for setting the update rate.

    Note: not all commands have an answer counterpart

  - The tag for a received data sentence is of the form TTDDD, where:
    * TT is the talker sending the data. The list of talkers is large but we are only interested in ones starting with a 'G':
      · GA - Galileo (Europe)
      · GB - BeiDou (China)
      · GI - NavIC (India)
      · GL - GLONASS (Russia)
      · GP - GPS (US)
      · GQ - QZSS (Japan)
      · GN - GNSS, a combination of the above
    * DDD is the data type of the sentence, this determines how to decode it. Again, the list of data types is long but we are only interested in a few:
      · RMC - Recommended Minimum Navigation Information
      · GLL - Geographic Position - Latitude/Longitude
      · GGA - Global Positioning System Fix Data
      · VTG - Track made good and Ground speed *(not currently parsed)*
      · ZDA - Time & Date - UTC, day, month, year and local time zone *(not currently parsed)*
      · GSA - GPS DOP and active satellites
      · GSV - Satellites in view
      · GRS - GPS Range Residuals *(not currently parsed)*
      · GST - GPS Pseudorange Noise Statistics *(not currently parsed)*
- DATA is separated from the TAG by a comma and is a comma separated list of data. Proprietary commands, and answers, will specify on their datasheet what the list of data is. The normal sentences generated by GPS modules are specified by NMEA. An unofficial list is here.
- '*' is the end of data delimiter.
- hh is the 1-byte checksum of all characters between '$' and '*' in hexadecimal.
- <CR><LF> is the mandatory sentence terminator

6.1.2 Checksums

When sending commands with the *send_command()* method it will add the necessary delimiters and calculate the checksum for you, eg.

```
gps.send_command(b'PMTK220,1000')
```

When receiving answers or data from the GPS module, if you use the *update()* method to poll the device it will reject any sentences with an invalid checksum and then try to parse the data. However, you can choose to manually pull data with the *read()* or *readline()* which will do no parsing or checksum validation.

6.1.3 Initial Configuration

```
import board
import busio
import adafruit_gps

USE_UART = True # Change this to False to connect via I2C

if USE_UART:
    # Create a serial connection for the GPS connection.
    uart = busio.UART(board.TX, board.RX, baudrate=9600, timeout=10)

    # for a computer, use the pyserial library for uart access
    # import serial
    # uart = serial.Serial("/dev/ttyUSB0", baudrate=9600, timeout=10)

    # Create a GPS module instance.
    gps = adafruit_gps.GPS(uart, debug=False)  # Use UART/pyserial
else:
    # If using I2C, we'll create an I2C interface to talk to using default pins
    i2c = board.I2C()

    # Create a GPS module instance.
    gps = adafruit_gps.GPS_GtopI2C(i2c, debug=False)  # Use I2C interface
```

6.1.4 Configuring the GPS

```
# Set update rate to 1000 milliseconds (1Hz)
gps.send_command(b"PMTK220,1000")

# Ask for specific data to be sent.
#                          A B C D E F G H                    I
gps.send_command(b'PMTK314,1,1,5,1,1,1,0,0,0,0,0,0,0,0,0,0,1,0')

#   A - send GLL sentences
#   B - send RMC sentences
#   C - send VTG sentences
#   D - send GGA sentences
#   E - send GSA sentences
#   F - send GSV sentences
#   G - send GRS sentences
#   H - send GST sentences
#   I - send ZDA sentences

# The number is how often to send the sentence compared to the update frequency.
# If the update frequency is 500ms and the number is 5, it will send that message
# every 2.5 seconds.
```

Note: Be aware that some data types send multiple sentences per update. So if you ask for 5 different types of data at 1Hz, you need to be able to handle at least 10 sentences per second. If the data is not read fast enough, the internal buffer and backlog behaviour is not specified.

6.1.5 Poll for data

```
while True:
    if gps.update():
        # A valid sentence was received - do something
        if gps.has_fix:
            print(f"{gps.latitude:.6f},{gps.longitude:.6f}")
        else:
            print("Waiting for a fix...")
    else:
        # No valid sentence was received, wait a moment.
        time.sleep(100)
```

The *update()* call takes care of reading data from the device and parsing it into usable data. This can then be accessed using the property accessors, eg. *has_fix*, *datetime*, latitude, longitude etc.

6.2 Selected Data Types

6.2.1 RMC - Recommended Minimum Navigation Information

```
       1         2 3       4 5        6 7   8   9   10  11 12
       |         | |       | |        | |   |   |    |   | |
$--RMC,hhmmss.ss,A,llll.ll,a,yyyyy.yy,a,x.x,x.x,xxxx,x.x,a*hh
$GNRMC,001031.00,A,4404.13993,N,12118.86023,W,0.146,,100117,,,A*7B
```

1. Time (UTC)
2. Status, A = Valid, V = Warning
3. Latitude
4. N or S
5. Longitude
6. E or W
7. Speed over ground, knots
8. Track made good, degrees true
9. Date, ddmmyy
10. Magnetic Variation, degrees
11. E or W
12. FAA mode indicator (NMEA 2.3 and later)
13. Checksum

6.2.2 GGA - Global Positioning System Fix Data

```
       1         2       3 4        5 6 7  8   9  10 11 12 13  14   15
       |         |       | |        | | |  |   |   | |   | |   |    |
$--GGA,hhmmss.ss,llll.ll,a,yyyyy.yy,a,x,xx,x.x,x.x,M,x.x,M,x.x,xxxx*hh
$GNGGA,001043.00,4404.14036,N,12118.85961,W,1,12,0.98,1113.0,M,-21.3,M,,*47
```

1. Time (UTC)
2. Latitude
3. N or S (North or South)
4. Longitude
5. E or W (East or West)
6. GPS Quality Indicator:
    0. Fix not available
    1. GPS fix
    2. Differential GPS fix
    3. PPS fix (values above 2 are NMEA 0183 v2.3 features)
    4. Real Time Kinematic
    5. Float RTK
    6. Estimated (dead reckoning)
    7. Manual input mode
    8. Simulation mode
7. Number of satellites in view, 00 - 12
8. Horizontal dilution of precision
9. Antenna altitude above/below mean-sea-level (geoid)
10. Units of antenna altitude, meters
11. Geoidal separation, the difference between the WGS-84 earth ellipsoid and mean-sea-level (geoid), "-" means mean-sea-level below ellipsoid
12. Units of geoidal separation, meters
13. Age of differential GPS data, time in seconds since last SC104 type 1 or 9 update, empty field when DGPS is not used
14. Differential reference station ID, 0000-1023
15. Checksum

Info about NMEA taken from here (2001). and here (2021)

6.3 Simple test

Ensure your device works with this simple test.

Listing 1: examples/gps_simpletest.py

```
# SPDX-FileCopyrightText: 2021 ladyada for Adafruit Industries
# SPDX-License-Identifier: MIT

# Simple GPS module demonstration.
# Will wait for a fix and print a message every second with the current location
# and other details.
```

(continues on next page)

(continued from previous page)

```
import time
import board
import busio

import adafruit_gps

# Create a serial connection for the GPS connection using default speed and
# a slightly higher timeout (GPS modules typically update once a second).
# These are the defaults you should use for the GPS FeatherWing.
# For other boards set RX = GPS module TX, and TX = GPS module RX pins.
uart = busio.UART(board.TX, board.RX, baudrate=9600, timeout=10)

# for a computer, use the pyserial library for uart access
# import serial
# uart = serial.Serial("/dev/ttyUSB0", baudrate=9600, timeout=10)

# If using I2C, we'll create an I2C interface to talk to using default pins
# i2c = board.I2C()

# Create a GPS module instance.
gps = adafruit_gps.GPS(uart, debug=False)  # Use UART/pyserial
# gps = adafruit_gps.GPS_GtopI2C(i2c, debug=False)  # Use I2C interface

# Initialize the GPS module by changing what data it sends and at what rate.
# These are NMEA extensions for PMTK_314_SET_NMEA_OUTPUT and
# PMTK_220_SET_NMEA_UPDATERATE but you can send anything from here to adjust
# the GPS module behavior:
#   https://cdn-shop.adafruit.com/datasheets/PMTK_A11.pdf

# Turn on the basic GGA and RMC info (what you typically want)
gps.send_command(b"PMTK314,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0")
# Turn on just minimum info (RMC only, location):
# gps.send_command(b'PMTK314,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0')
# Turn off everything:
# gps.send_command(b'PMTK314,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0')
# Turn on everything (not all of it is parsed!)
# gps.send_command(b'PMTK314,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0')

# Set update rate to once a second (1hz) which is what you typically want.
gps.send_command(b"PMTK220,1000")
# Or decrease to once every two seconds by doubling the millisecond value.
# Be sure to also increase your UART timeout above!
# gps.send_command(b'PMTK220,2000')
# You can also speed up the rate, but don't go too fast or else you can lose
# data during parsing.  This would be twice a second (2hz, 500ms delay):
# gps.send_command(b'PMTK220,500')

# Main loop runs forever printing the location, etc. every second.
last_print = time.monotonic()
while True:
    # Make sure to call gps.update() every loop iteration and at least twice
    # as fast as data comes from the GPS unit (usually every second).
    # This returns a bool that's true if it parsed new data (you can ignore it
    # though if you don't care and instead look at the has_fix property).
    gps.update()
    # Every second print out current location details if there's a fix.
    current = time.monotonic()
```

(continues on next page)

(continued from previous page)

```
    if current - last_print >= 1.0:
        last_print = current
        if not gps.has_fix:
            # Try again if we don't have a fix yet.
            print("Waiting for fix...")
            continue
        # We have a fix! (gps.has_fix is true)
        # Print out details about the fix like location, date, etc.
        print("=" * 40)  # Print a separator line.
        print(
            "Fix timestamp: {}/{}/{} {:02}:{:02}:{:02}".format(
                gps.timestamp_utc.tm_mon,  # Grab parts of the time from the
                gps.timestamp_utc.tm_mday,  # struct_time object that holds
                gps.timestamp_utc.tm_year,  # the fix time.  Note you might
                gps.timestamp_utc.tm_hour,  # not get all data like year, day,
                gps.timestamp_utc.tm_min,  # month!
                gps.timestamp_utc.tm_sec,
            )
        )
        print("Latitude: {0:.6f} degrees".format(gps.latitude))
        print("Longitude: {0:.6f} degrees".format(gps.longitude))
        print("Fix quality: {}".format(gps.fix_quality))
        # Some attributes beyond latitude, longitude and timestamp are optional
        # and might not be present.  Check if they're None before trying to use!
        if gps.satellites is not None:
            print("# satellites: {}".format(gps.satellites))
        if gps.altitude_m is not None:
            print("Altitude: {} meters".format(gps.altitude_m))
        if gps.speed_knots is not None:
            print("Speed: {} knots".format(gps.speed_knots))
        if gps.track_angle_deg is not None:
            print("Track angle: {} degrees".format(gps.track_angle_deg))
        if gps.horizontal_dilution is not None:
            print("Horizontal dilution: {}".format(gps.horizontal_dilution))
        if gps.height_geoid is not None:
            print("Height geoid: {} meters".format(gps.height_geoid))
```

6.4 Echo test

Simple GPS module demonstration. This will print NMEA sentences received from the GPS, great for testing connection. This uses the GPS to send some commands, then reads directly from the GPS.

Listing 2: examples/gps_echotest.py

```
# SPDX-FileCopyrightText: 2021 ladyada for Adafruit Industries
# SPDX-License-Identifier: MIT

# Simple GPS module demonstration.
# Will print NMEA sentences received from the GPS, great for testing connection
# Uses the GPS to send some commands, then reads directly from the GPS
import time
import board
import busio

```

(continues on next page)

(continued from previous page)

```
import adafruit_gps

# Create a serial connection for the GPS connection using default speed and
# a slightly higher timeout (GPS modules typically update once a second).
# These are the defaults you should use for the GPS FeatherWing.
# For other boards set RX = GPS module TX, and TX = GPS module RX pins.
uart = busio.UART(board.TX, board.RX, baudrate=9600, timeout=10)

# for a computer, use the pyserial library for uart access
# import serial
# uart = serial.Serial("/dev/ttyUSB0", baudrate=9600, timeout=10)

# If using I2C, we'll create an I2C interface to talk to using default pins
# i2c = board.I2C()

# Create a GPS module instance.
gps = adafruit_gps.GPS(uart)  # Use UART/pyserial
# gps = adafruit_gps.GPS_GtopI2C(i2c)  # Use I2C interface

# Initialize the GPS module by changing what data it sends and at what rate.
# These are NMEA extensions for PMTK_314_SET_NMEA_OUTPUT and
# PMTK_220_SET_NMEA_UPDATERATE but you can send anything from here to adjust
# the GPS module behavior:
#   https://cdn-shop.adafruit.com/datasheets/PMTK_A11.pdf

# Turn on the basic GGA and RMC info (what you typically want)
gps.send_command(b"PMTK314,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0")
# Turn on just minimum info (RMC only, location):
# gps.send_command(b'PMTK314,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0')
# Turn off everything:
# gps.send_command(b'PMTK314,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0')
# Tuen on everything (not all of it is parsed!)
# gps.send_command(b'PMTK314,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0')

# Set update rate to once a second (1hz) which is what you typically want.
gps.send_command(b"PMTK220,1000")
# Or decrease to once every two seconds by doubling the millisecond value.
# Be sure to also increase your UART timeout above!
# gps.send_command(b'PMTK220,2000')
# You can also speed up the rate, but don't go too fast or else you can lose
# data during parsing.  This would be twice a second (2hz, 500ms delay):
# gps.send_command(b'PMTK220,500')

# Main loop runs forever printing data as it comes in
timestamp = time.monotonic()
while True:
    data = gps.read(32)  # read up to 32 bytes
    # print(data)  # this is a bytearray type

    if data is not None:
        # convert bytearray to string
        data_string = "".join([chr(b) for b in data])
        print(data_string, end="")

    if time.monotonic() - timestamp > 5:
        # every 5 seconds...
        gps.send_command(b"PMTK605")  # request firmware version
```

(continues on next page)

(continued from previous page)

```
        timestamp = time.monotonic()
```

6.5 Time source

Simple script using GPS timestamps as RTC time source. The GPS timestamps are available without a full location fix if a single satellite can be seen. The GPS unit will keep the track of time while there is power source (i.e. a coin cell battery.)

Listing 3: examples/gps_time_source.py

```
# SPDX-FileCopyrightText: 2021 ladyada for Adafruit Industries
# SPDX-License-Identifier: MIT

# Simple script using GPS timestamps as RTC time source
# The GPS timestamps are available without a fix and keep the track of
# time while there is powersource (ie coin cell battery)

import time
import board
import busio
import rtc
import adafruit_gps

uart = busio.UART(board.TX, board.RX, baudrate=9600, timeout=10)
# i2c = busio.I2C(board.SCL, board.SDA)

gps = adafruit_gps.GPS(uart, debug=False)
# gps = adafruit_gps.GPS_GtopI2C(i2c, debug=False)  # Use I2C interface

gps.send_command(b"PMTK314,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0")
gps.send_command(b"PMTK220,1000")

print("Set GPS as time source")
rtc.set_time_source(gps)
the_rtc = rtc.RTC()


def _format_datetime(datetime):
    return "{:02}/{:02}/{} {:02}:{:02}:{:02}".format(
        datetime.tm_mon,
        datetime.tm_mday,
        datetime.tm_year,
        datetime.tm_hour,
        datetime.tm_min,
        datetime.tm_sec,
    )


last_print = time.monotonic()
while True:

    gps.update()
    # Every second print out current time from GPS, RTC and time.localtime()
    current = time.monotonic()
```

(continues on next page)

(continued from previous page)

```
    if current - last_print >= 1.0:
        last_print = current
        if not gps.timestamp_utc:
            print("No time data from GPS yet")
            continue
        # Time & date from GPS informations
        print("Fix timestamp: {}".format(_format_datetime(gps.timestamp_utc)))

        # Time & date from internal RTC
        print("RTC timestamp: {}".format(_format_datetime(the_rtc.datetime)))

        # Time & date from time.localtime() function
        local_time = time.localtime()

        print("Local time: {}".format(_format_datetime(local_time)))
```

6.6 Data logging

Simple GPS datalogging demonstration. This example uses the GPS library and to read raw NMEA sentences over I2C or UART from the GPS unit and dumps them to a file on an SD card (recommended), microcontroller internal storage (be careful as only a few kilobytes are available), or to a filesystem.

If you are using a microcontroller, before writing to internal storage you MUST carefully follow the steps in this guide to enable writes to the internal filesystem: Writing to the filesystem

Listing 4: examples/gps_datalogging.py

```
# SPDX-FileCopyrightText: 2021 ladyada for Adafruit Industries
# SPDX-License-Identifier: MIT

# Simple GPS datalogging demonstration.
# This example uses the GPS library and to read raw NMEA sentences
# over I2C or UART from the GPS unit and dumps them to a file on an SD card
# (recommended), microcontroller internal storage (be careful as only a few
# kilobytes are available), or to a filesystem.
# If you are using a microcontroller, before writing to internal storage you
#  MUST carefully follow the steps in this guide to enable writes to the
# internal filesystem:
#  https://learn.adafruit.com/adafruit-ultimate-gps-featherwing/circuitpython-library
import board
import busio
import adafruit_gps

# Path to the file to log GPS data.  By default this will be appended to
# which means new lines are added at the end and all old data is kept.
# Change this path to point at internal storage (like '/gps.txt') or SD
# card mounted storage ('/sd/gps.txt') as desired.
LOG_FILE = "gps.txt"  # Example for writing to internal path gps.txt

# File more for opening the log file.  Mode 'ab' means append or add new lines
# to the end of the file rather than erasing it and starting over.  If you'd
# like to erase the file and start clean each time use the value 'wb' instead.
LOG_MODE = "ab"

```

(continues on next page)

(continued from previous page)

```
# If writing to SD card on a microcontroller customize and uncomment these
# lines to import the necessary library and initialize the SD card:
# NOT for use with a single board computer like Raspberry Pi!
"""
import adafruit_sdcard
import digitalio
import storage

SD_CS_PIN = board.D10  # CS for SD card using Adalogger Featherwing
spi = busio.SPI(board.SCK, MOSI=board.MOSI, MISO=board.MISO)
sd_cs = digitalio.DigitalInOut(SD_CS_PIN)
sdcard = adafruit_sdcard.SDCard(spi, sd_cs)
vfs = storage.VfsFat(sdcard)
storage.mount(vfs, '/sd')     # Mount SD card under '/sd' path in filesystem.
LOG_FILE = '/sd/gps.txt'      # Example for writing to SD card path /sd/gps.txt
"""

# Create a serial connection for the GPS connection using default speed and
# a slightly higher timeout (GPS modules typically update once a second).
# These are the defaults you should use for the GPS FeatherWing.
# For other boards set RX = GPS module TX, and TX = GPS module RX pins.
uart = busio.UART(board.TX, board.RX, baudrate=9600, timeout=10)

# If using a USB/Serial converter, use pyserial and update the serial
# port name to match the serial connection for the GPS!
# import serial
# uart = serial.Serial("/dev/ttyUSB0", baudrate=9600, timeout=10)

# If using I2C, we'll create an I2C interface to talk to using default pins
# i2c = board.I2C()

# Create a GPS module instance.
gps = adafruit_gps.GPS(uart)  # Use UART/pyserial
# gps = adafruit_gps.GPS_GtopI2C(i2c)  # Use I2C interface

# Main loop just reads data from the GPS module and writes it back out to
# the output file while also printing to serial output.
with open(LOG_FILE, LOG_MODE) as outfile:
    while True:
        sentence = gps.readline()
        if not sentence:
            continue
        print(str(sentence, "ascii").strip())
        outfile.write(sentence)
        outfile.flush()
```

6.7 Satellite fix

This example uses GSA and GSV sentences from the GPS device to report on the quality of the received data from the satellites.

- GSA - DOP(Dilution of Precision) and active satellites
- GSV - Satellites in view

Listing 5: examples/gps_satellitefix.py

```
# SPDX-FileCopyrightText: 2021 lesamouraipourpre
# SPDX-License-Identifier: MIT

# This example uses GSA and GSV sentences from the GPS device to report on the
# quality of the received data from the satellites.
# * GSA - DOP(Dilution of Precision) and active satellites
# * GSV - Satellites in view

import time
import board

import adafruit_gps

# Create a serial connection for the GPS connection using default speed and
# a slightly higher timeout (GPS modules typically update once a second).
# These are the defaults you should use for the GPS FeatherWing.
# For other boards set RX = GPS module TX, and TX = GPS module RX pins.
# uart = busio.UART(board.TX, board.RX, baudrate=9600, timeout=10)

# for a computer, use the pyserial library for uart access
# import serial
# uart = serial.Serial("/dev/ttyUSB0", baudrate=9600, timeout=10)

# If using I2C, we'll create an I2C interface to talk to using default pins
i2c = board.I2C()

# Create a GPS module instance.
# gps = adafruit_gps.GPS(uart, debug=False)  # Use UART/pyserial
gps = adafruit_gps.GPS_GtopI2C(i2c, debug=False)  # Use I2C interface

# Initialize the GPS module by changing what data it sends and at what rate.
# These are NMEA extensions for PMTK_314_SET_NMEA_OUTPUT and
# PMTK_220_SET_NMEA_UPDATERATE but you can send anything from here to adjust
# the GPS module behavior:
#   https://cdn-shop.adafruit.com/datasheets/PMTK_A11.pdf

# Turn on everything (not all of it is parsed!)
gps.send_command(b"PMTK314,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0")

# Set update rate to once a second (1hz) which is what you typically want.
gps.send_command(b"PMTK220,1000")
# Or decrease to once every two seconds by doubling the millisecond value.
# Be sure to also increase your UART timeout above!
# gps.send_command(b'PMTK220,2000')
# You can also speed up the rate, but don't go too fast or else you can lose
# data during parsing.  This would be twice a second (2hz, 500ms delay):
# gps.send_command(b'PMTK220,500')


def format_dop(dop):
    # https://en.wikipedia.org/wiki/Dilution_of_precision_(navigation)
    if dop > 20:
        msg = "Poor"
    elif dop > 10:
        msg = "Fair"
```

(continues on next page)

(continued from previous page)

```
    elif dop > 5:
        msg = "Moderate"
    elif dop > 2:
        msg = "Good"
    elif dop > 1:
        msg = "Excellent"
    else:
        msg = "Ideal"
    return f"{dop} - {msg}"


talkers = {
    "GA": "Galileo",
    "GB": "BeiDou",
    "GI": "NavIC",
    "GL": "GLONASS",
    "GP": "GPS",
    "GQ": "QZSS",
    "GN": "GNSS",
}

# Main loop runs forever printing the location, etc. every second.
last_print = time.monotonic()
while True:
    # Make sure to call gps.update() every loop iteration and at least twice
    # as fast as data comes from the GPS unit (usually every second).
    # This returns a bool that's true if it parsed new data (you can ignore it
    # though if you don't care and instead look at the has_fix property).
    if not gps.update() or not gps.has_fix:
        time.sleep(0.1)
        continue

    if gps.nmea_sentence[3:6] == "GSA":
        print(f"{gps.latitude:.6f}, {gps.longitude:.6f} {gps.altitude_m}m")
        print(f"2D Fix: {gps.has_fix}  3D Fix: {gps.has_3d_fix}")
        print(f"  PDOP (Position Dilution of Precision): {format_dop(gps.pdop)}")
        print(f"  HDOP (Horizontal Dilution of Precision): {format_dop(gps.hdop)}")
        print(f"  VDOP (Vertical Dilution of Precision): {format_dop(gps.vdop)}")
        print("Satellites used for fix:")
        for s in gps.sat_prns:
            talker = talkers[s[0:2]]
            number = s[2:]
            print(f"  {talker}-{number} ", end="")
            if gps.sats is None:
                print("- no info")
            else:
                try:
                    sat = gps.sats[s]
                    if sat is None:
                        print("- no info")
                    else:
                        print(f"Elevation:{sat[1]}* Azimuth:{sat[2]}* SNR:{sat[3]}dB")
                except KeyError:
                    print("- no info")
        print()
```

6.8 adafruit_gps

GPS parsing module. Can parse simple NMEA data sentences from serial GPS modules to read latitude, longitude, and more.

- Author(s): Tony DiCola, James Carr

6.8.1 Implementation Notes

Hardware:

- Adafruit Ultimate GPS Breakout
- Adafruit Ultimate GPS FeatherWing

Software and Dependencies:

- Adafruit CircuitPython firmware for the ESP8622 and M0-based boards: https://github.com/adafruit/circuitpython/releases

class adafruit_gps.**GPS**(*uart*, *debug=False*)

GPS parsing module. Can parse simple NMEA data sentences from serial GPS modules to read latitude, longitude, and more.

datetime

Return struct_time object to feed rtc.set_time_source() function

has_3d_fix

Returns true if there is a 3d fix available. use has_fix to determine if a 2d fix is available, passing it the same data

has_fix

True if a current fix for location information is available.

in_waiting

Returns number of bytes available in UART read buffer

nmea_sentence

Return raw_sentence which is the raw NMEA sentence read from the GPS

read(*num_bytes*)

Read up to num_bytes of data from the GPS directly, without parsing. Returns a bytearray with up to num_bytes or None if nothing was read

readline()

Returns a newline terminated bytearray, must have timeout set for the underlying UART or this will block forever!

send_command(*command*, *add_checksum=True*)

Send a command string to the GPS. If add_checksum is True (the default) a NMEA checksum will automatically be computed and added. Note you should NOT add the leading $ and trailing * to the command as they will automatically be added!

update()

Check for updated data from the GPS module and process it accordingly. Returns True if new data was processed, and False if nothing new was received.

write(*bytestr*)

Write a bytestring data to the GPS directly, without parsing or checksums

class adafruit_gps.**GPS_GtopI2C**(*i2c_bus*, ***, *address=16*, *debug=False*, *timeout=5*)

GTop-compatible I2C GPS parsing module. Can parse simple NMEA data sentences from an I2C-capable GPS module to read latitude, longitude, and more.

in_waiting

Returns number of bytes available in UART read buffer, always 16 since I2C does not have the ability to know how much data is available

read(*num_bytes=1*)

Read up to num_bytes of data from the GPS directly, without parsing. Returns a bytearray with up to num_bytes or None if nothing was read

readline()

Returns a newline terminated bytearray, must have timeout set for the underlying UART or this will block forever!

write(*bytestr*)

Write a bytestring data to the GPS directly, without parsing or checksums

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Python Module Index

a

Index