Adafruit GPS Library Documentation

Release 1.0

Tony DiCola

Sep 16, 2019

Contents

GPS parsing module. Can parse simple NMEA data sentences from serial 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

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 3

About NMEA Data

This GPS module uses the NMEA 0183 protocol.

This data is formatted by the GPS in one of two ways.

The first of these is GGA. GGA has more or less everything you need.

Here's an explanation of GGA:

```
                                                      11
       1          2       3 4        5 6 7  8   9  10 |  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
```

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
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, null field when DGPS is not used
14. Differential reference station ID, 0000-1023
15. Checksum

The second of these is RMC. RMC is Recommended Minimum Navigation Information.

Here's an explanation of RMC:

```
                                                          12
       1          2 3        4 5         6 7   8    9    10   11|
       |          | |        | |         | |   |    |     |    | |
$--RMC,hhmmss.ss,A,llll.ll,a,yyyyy.yy,a,x.x,x.x,xxxx,x.x,a*hh
```

1. Time (UTC)
2. Status, V = Navigation receiver 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. Checksum

Info about NMEA taken from here.

CHAPTER 4

Contributing

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

CHAPTER 5

Building locally

To build this library locally, you'll need to install the circuitpython-build-tools package.

```
python3 -m venv .env
source .env/bin/activate
pip install circuitpython-build-tools
```

Once installed, make sure you are in the virtual environment:

```
source .env/bin/activate
```

Then run the build:

```
circuitpython-build-bundles --filename_prefix adafruit-circuitpython-gps --library_
↪location .
```

5.1 Sphinx documentation

Sphinx is used to build the documentation based on rST files and comments in the code. First, install dependencies (feel free to reuse the virtual environment from above):

```
python3 -m venv .env
source .env/bin/activate
pip install Sphinx sphinx-rtd-theme
```

Now, once you have the virtual environment activated:

```
cd docs
sphinx-build -E -W -b html . _build/html
```

This will output the documentation to `docs/_build/html`. Open the index.html in your browser to view them. It will also (due to -W) error out on any warning like Travis will. This is a good way to locally verify it will pass.

CHAPTER 6

Table of Contents

6.1 Simple test

Ensure your device works with this simple test.

Listing 1: examples/gps_simpletest.py

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

import adafruit_gps


# Define RX and TX pins for the board's serial port connected to the GPS.
# 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.
RX = board.RX
TX = board.TX

# Create a serial connection for the GPS connection using default speed and
# a slightly higher timeout (GPS modules typically update once a second).
uart = busio.UART(TX, RX, baudrate=9600, timeout=30)

# for a computer, use the pyserial library for uart access
#import serial
#uart = serial.Serial("/dev/ttyUSB0", baudrate=9600, timeout=3000)

# Create a GPS module instance.
gps = adafruit_gps.GPS(uart, debug=False)

```

(continues on next page)

(continued from previous page)

```
# 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 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()
    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))
```

(continues on next page)

(continued from previous page)

```
85          if gps.altitude_m is not None:
86              print('Altitude: {} meters'.format(gps.altitude_m))
87          if gps.speed_knots is not None:
88              print('Speed: {} knots'.format(gps.speed_knots))
89          if gps.track_angle_deg is not None:
90              print('Track angle: {} degrees'.format(gps.track_angle_deg))
91          if gps.horizontal_dilution is not None:
92              print('Horizontal dilution: {}'.format(gps.horizontal_dilution))
93          if gps.height_geoid is not None:
94              print('Height geo ID: {} meters'.format(gps.height_geoid))
```

6.2 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

6.2.1 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.

nmea_sentence

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

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.

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Python Module Index

a

Index