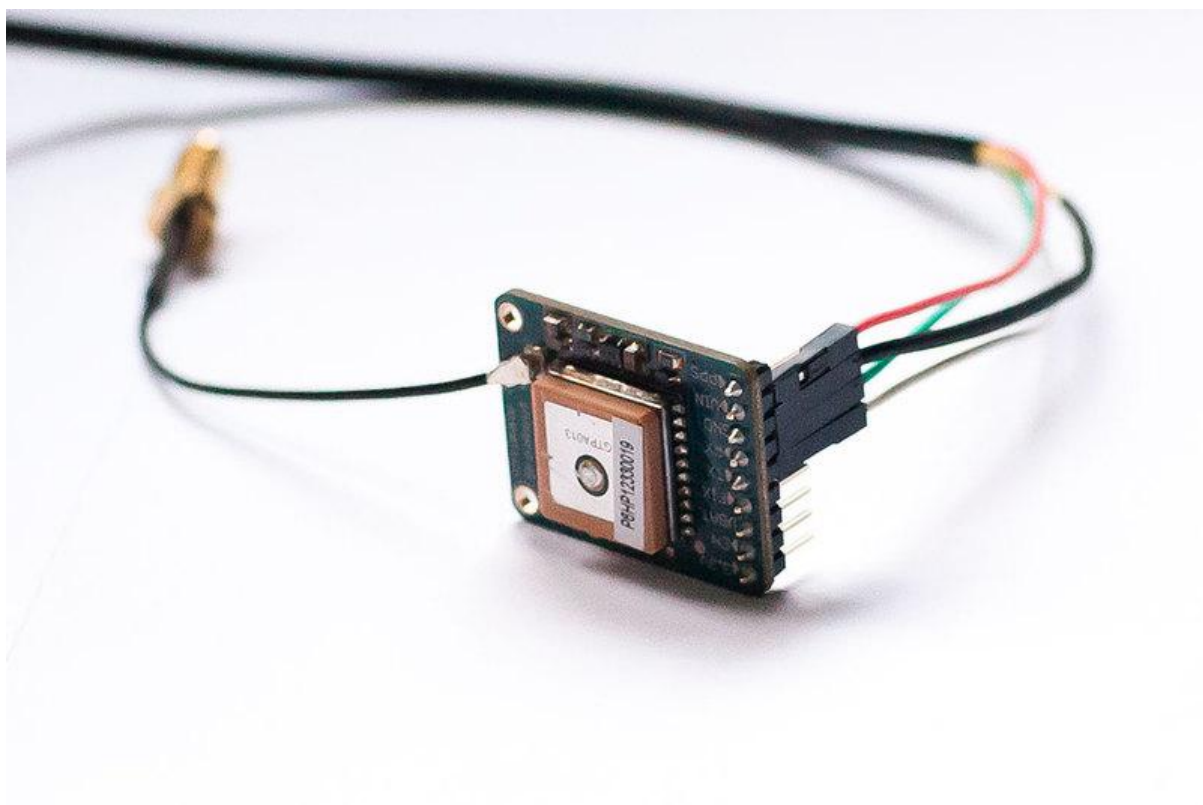




Adafruit Ultimate GPS with gpsd

Created by Kevin Townsend



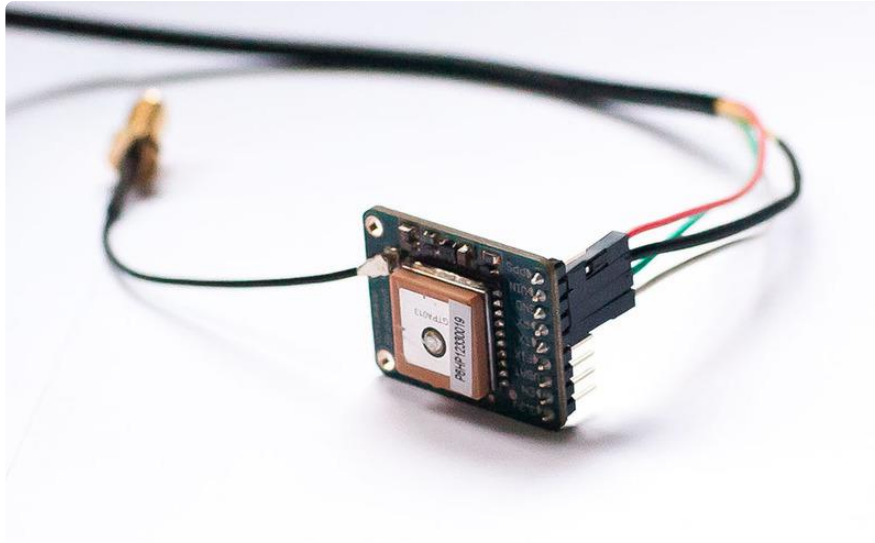
<https://learn.adafruit.com/adafruit-ultimate-gps-on-the-raspberry-pi>

Last updated on 2022-12-01 01:55:29 PM EST

Table of Contents

Introduction	3
<hr/>	
• What you'll need:	
Setting Everything Up	4
<hr/>	
• Hooking The Breakout Up	
• Setting up the USB Adapter	
• Installing a GPS Daemon (gpsd)	
• Raspbian Jessie systemd service fix	
• Testing it Out	
Using your GPS	7
<hr/>	
Using UART instead of USB	9
<hr/>	
• Disable Serial console and Enable UART	
• Reboot your Pi	
• Restart GPSD with HW UART	
Further Resources	12
<hr/>	

Introduction



How easy is it to get your Raspberry Pi eavesdropping on satellites 20,000 km up in the sky? Wonderfully easy thanks to Linux, and affordable thanks to Adafruit's [Ultimate GPS Breakout \(http://adafru.it/746\)](http://adafru.it/746)!

This quick learning guide will show you everything you need to do to add position tracking to your Pi project using the open source GPS daemon 'gpsd' and an inexpensive [USB to TTL adapter cable \(http://adafru.it/954\)](http://adafru.it/954) or via direct-wiring to the built-in Pi UART pins

Please note this guide installs a system service called gpsd which you can then query for data. You may be better off just using pure python to read data from the GPS, its less complex in many cases

If you don't need gpsd, click below to follow the pure-python GPS tutorial

Using the Ultimate GPS directly with
Python / CircuitPython

What you'll need:

- [A Raspberry Pi \(http://adafru.it/998\)](http://adafru.it/998)

- An [Ultimate GPS Breakout \(http://adafru.it/746\)](http://adafru.it/746)
- A [USB to TTL Adapter \(http://adafru.it/954\)](http://adafru.it/954) Cable (or something compatible)

[Don't forget to also read our Ultimate GPS tutorial which has a lot of information about this GPS module and datasheets/example code that you will find handy! \(\)](#)

Setting Everything Up

The easiest way to get started is to use an inexpensive [USB to TTL adapter cable \(http://adafru.it/954\)](http://adafru.it/954) with your GPS module.

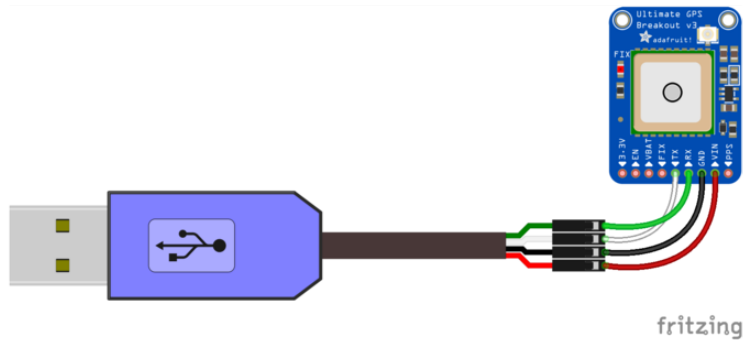
You can of course use the HW UART directly on the Pi, but as you can see in this tutorial ([Freeing UART on the Pi \(\)](#)) it's a bit more complicated, and there are no secondary consequences with the USB adapter.

This tutorial will assume that we are using the USB to TTL cable mentioned above, and that we are running on Raspbian. Raspbian already has the drivers for PL2303-based cables pre-installed, so you just need to plug it in and it should show up as /dev/ttyUSB0).

Hooking The Breakout Up

The first thing you'll need to do is to hook your Ultimate GPS Breakout up to the Pi with the adapter cable. The following diagram shows you what you need to know, essentially just connecting the cables of the same color together.

- GPS Vin to USB 5V or 3V (red wire on USB console cable)
- GPS Ground to USB Ground (black wire)
- GPS RX to USB TX (green wire)
- GPS TX to USB RX (white wire)



While the module on the Ultimate GPS Breakout has an exceptionally sensitive antenna and may work indoors as is, you may want to pick up an [external GPS Antenna](http://adafru.it/960) (<http://adafru.it/960>) and an [SMA to u.FL adapter cable](http://adafru.it/851) (<http://adafru.it/851>) if this is for indoor use. This will allow you to keep the Pi and GPS breakout indoors, but run the antenna out a window or at least near one for improved reliability and signal integrity.

Setting up the USB Adapter

Once you plug the USB cable into the Pi, the adapter should show up as `/dev/ttyUSB0` (though the '0' may be different if you have other `ttyUSB` adapters present).

You can see a list of all `ttyUSB` devices by entering the following into the console (I'm using the 'terminal' feature in Adafruit's browser-based WebIDE here for convenience sake!):

```
ls /dev/ttyUSB*
```

```
webide@raspberrypi /usr/share/adafruit/webide/repositories $ ls /dev/ttyUSB*  
/dev/ttyUSB0
```

If you have any problems, you can enter the following command to see the USB devices on your Pi:

```
sudo lsusb
```

Which should show you the USB adapter (Prolific PL2303), as follows:

```
webide@raspberrypi /usr/share/adafruit/webide/repositories $ sudo lsusb  
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub  
Bus 001 Device 002: ID 0424:9512 Standard Microsystems Corp.  
Bus 001 Device 003: ID 0424:ec00 Standard Microsystems Corp.  
Bus 001 Device 005: ID 067b:2303 Prolific Technology, Inc. PL2303 Serial Port
```

If you just want to do a quick check to see what data is coming out of the GPS, you can enter the following command, followed by Ctrl-C to quit:

```
sudo cat /dev/ttyUSB0
```

Installing a GPS Daemon (gpsd)

The next step is installing some software on your Raspberry Pi that understands the serial data that your GPS module is providing via `/dev/ttyUSB0`.

Thankfully other people have already done all the hard work for you of properly parsing the raw GPS data, and we can use (amongst other options) a nice little package named 'gpsd', which essentially acts as a layer between your applications and the actual GPS hardware, gracefully handling parsing errors, and providing a common, well-defined interfaces to any GPS module.

To install gpsd, simply run the following commands from the console:

```
sudo apt-get install gpsd gpsd-clients
```

... which will install the required packages (an internet connection will be required for this step!)

Raspbian Jessie systemd service fix

Note if you're using the Raspbian Jessie or later release you'll need to disable a systemd service that gpsd installs. This service has systemd listen on a local socket and run gpsd when clients connect to it, however it will also interfere with other gpsd instances that are manually run (like in this guide). You will need to disable the gpsd systemd service by running the following commands:

```
sudo systemctl stop gpsd.socket  
sudo systemctl disable gpsd.socket
```

Should you ever want to enable the default gpsd systemd service you can run these commands to restore it (but remember the rest of the steps in this guide won't work!):

```
sudo systemctl enable gpsd.socket  
sudo systemctl start gpsd.socket
```

After disabling the gpsd systemd service above you're ready to try running gpsd manually. Now run the following command to manually start gpsd and point it at the GPS breakout on the USB serial adapter port:

```
sudo gpsd /dev/ttyUSB0 -F /var/run/gpsd.sock
```

... which will point the gps daemon to our GPS device on the USB to TTY adapter cable (simply substitute '/dev/ttyUSB0' for another destination if required).

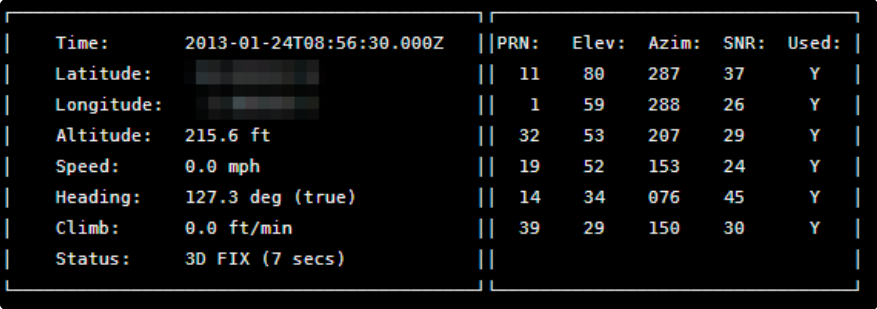
Testing it Out

After a few seconds, gpsd should open up the proper socket and if the GPS is locked we should be able to get some data from the GPS module.

To test this, we can use the following command:

```
cgps -s
```

If you have a fix, you'll see something like the following information in the terminal window:

A terminal window with a black background and white text. It displays the output of the 'cgps -s' command. The output is split into two columns. The left column shows various GPS data points, and the right column shows a table of PRN, Elevation, Azimuth, SNR, and Usage for several satellites.

Time:	2013-01-24T08:56:30.000Z	PRN:	Elev:	Azim:	SNR:	Used:
Latitude:	[REDACTED]	11	80	287	37	Y
Longitude:	[REDACTED]	1	59	288	26	Y
Altitude:	215.6 ft	32	53	207	29	Y
Speed:	0.0 mph	19	52	153	24	Y
Heading:	127.3 deg (true)	14	34	076	45	Y
Climb:	0.0 ft/min	39	29	150	30	Y
Status:	3D FIX (7 secs)					

If you have any problems and cgps always displays 'NO FIX' under status and then aborts after a few seconds, you may need to restart the gpsd service. You can do that via the following commands:

```
sudo killall gpsd  
sudo gpsd /dev/ttyUSB0 -F /var/run/gpsd.sock
```

Using your GPS

Now that your GPS is up and running, and gpsd is playing nice with it, it's time to do something with the data!

The easiest way to get started is using a bit of Python code to access gpsd. First, install the `gps` module from PyPI:

```
pip3 install gps
```

Next, use `nano` or the text editor of your choice to save the following code in a file called `gpstest.py`:

```
import gps

# Listen on port 2947 (gpsd) of localhost
session = gps.gps("localhost", "2947")
session.stream(gps.WATCH_ENABLE | gps.WATCH_NEWSTYLE)

while True:
    try:
        report = session.next()
        # Wait for a 'TPV' report and display the current time
        # To see all report data, uncomment the line below
        # print(report)
        if report['class'] == 'TPV':
            if hasattr(report, 'time'):
                print(report.time)
    except KeyError:
        pass
    except KeyboardInterrupt:
        quit()
    except StopIteration:
        session = None
        print("GPSD has terminated")
```

This should give you something similar to the following (with an update every second or so):

```
pi@raspberrypi:~$ python3 gpstest.py
2018-10-01T20:18:09.000Z
2018-10-01T20:18:10.000Z
2018-10-01T20:18:11.000Z
2018-10-01T20:18:12.000Z
2018-10-01T20:18:13.000Z
```

Looking for position data rather than just the timestamp? Essentially, all you have to do is parse the 'report' data following the example above.

To see what data is available, you can uncomment the `print(report)` line, and then just look at the different values and class names and pick and choose whatever you want.

```
pi@raspberrypi:~$ python3 gpstest.py
<dictwrapper: {'proto_major': 3, 'release': '3.16', 'rev': '3.16-4', 'class': 'VERSION', 'proto_minor': 11}>
<dictwrapper: {'devices': [{'subtype': 'ANL_2_31_3330_13101700-5632', 'bps': 9600, 'minicycle': 0.2, 'parity': 'N', 'activated': '2018-10-01T21:09:23.99
52', 'stopbits': 1, 'class': 'DEVICE', 'native': 0, 'path': '/dev/ttyUSB0', 'flags': 1, 'driver': 'MKK-3301', 'cycle': 1.0}], 'class': 'DEVICES'}>
<dictwrapper: {'enable': True, 'raw': 0, 'nmea': False, 'timing': False, 'json': True, 'pps': False, 'split24': False, 'class': 'WATCH', 'scaled': Fals
e}>
<dictwrapper: {'track': 37.59, 'climb': 0.326, 'alt': 1634.1, 'eps': 0.03, 'time': '2018-10-01T21:09:24.000Z', 'speed': 0.051, 'epd': 2.6488, 'epv': 5.
277, 'lon': 1.77, 'mode': 3, 'epv': 17.488, 'ept': 0.005, 'epc': 0.01, 'lat': 5.804, 'epx': 5.804, 'device': '/dev/ttyUSB0', 'class':
'TPV'}>
2018-10-01T21:09:24.000Z
<dictwrapper: {'track': 70.17, 'climb': 0.0, 'alt': 1634.1, 'eps': 34.98, 'time': '2018-10-01T21:09:25.000Z', 'speed': 0.041, 'epv': 5.175, 'lon': 1.77, 'mode': 3, 'epv': 17.488, 'ept': 0.005, 'epc': 10.75, 'lat': 5.804, 'epx': 5.804, 'device': '/dev/ttyUSB0', 'class': 'TPV'}>
2018-10-01T21:09:25.000Z
<dictwrapper: {'tdop': 6.17, 'hdop': 1.18, 'gdop': 13.67, 'vdop': 0.9, 'pdop': 1.48, 'ydop': 4.66, 'satellites': [<dictwrapper: {'ss': 39, 'used': True, 'PRN': 20, 'el': 73, 'az': 358}>, <dictwrapper: {'ss': 18, 'used': True, 'PRN': 21, 'el': 69, 'az': 133}>, <dictwrapper: {'ss': 32, 'used': True, 'PRN': 10, 'el': 56, 'az': 294}>, <dictwrapper: {'ss': 28, 'used': True, 'PRN': 27, 'el': 36, 'az': 291}>, <dictwrapper: {'ss': 24, 'used': True, 'PRN': 15, 'el': 35, 'az': 513}>, <dictwrapper: {'ss': 30, 'used': False, 'PRN': 135, 'el': 35, 'az': 219}>, <dictwrapper: {'ss': 33, 'used': True, 'PRN': 32, 'el': 30, 'az': 212}>, <dictwrapper: {'ss': 30, 'used': True, 'PRN': 24, 'el': 29, 'az': 96}>, <dictwrapper: {'ss': 17, 'used': False, 'PRN': 8, 'el': 15, 'az': 319}>, <dictwrapper: {'ss': 20, 'used': False, 'PRN': 14, 'el': 9, 'az': 212}>, <dictwrapper: {'ss': 0, 'used': False, 'PRN': 13, 'el': 1, 'az': 30}>], 'device': '/dev/ttyUSB0', 'class': 'SKV', 'hdop': 1.55}>
```


For example, you could use the following code to get the current speed using the TPV class:

```
if report['class'] == 'TPV':
    if hasattr(report, 'speed'):
        print(report.speed * gps.MPS_TO_KPH)
```

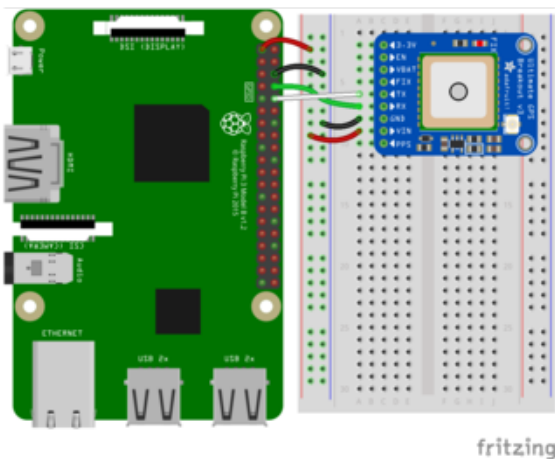
That's it! It's pretty painless, and now it's up to you to figure out what you want to do with you latitude, longitude, date and time, speed, altitude, etc.!

Using UART instead of USB

If you wish to use HW UART instead of the USB cable, it's perfectly possible - you just need to do a bit more work to free the UART up on your Pi.

To get started, hook the GPS module up to your Pi as follows, cross-connecting the TX and RX pins (TX on one device goes to RX on the other and vice versa), and supply 5V from the Pi to the VIN pin on the GPS module:

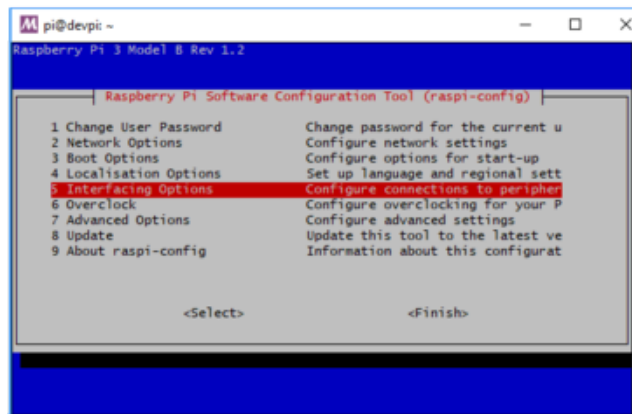
We designed the Ultimate GPS with a built-in regulator, so even if it's powered with 5V, the signal levels are still 3.3V - safe for your Pi!



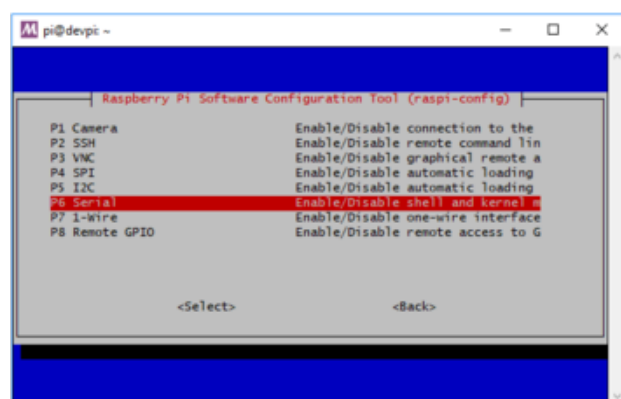
GPS Vin to 3.3V (red wire)
GPS Ground to Ground (black wire)
GPS RX to TX (green wire)
GPS TX to RX (white wire)

Disable Serial console and Enable UART

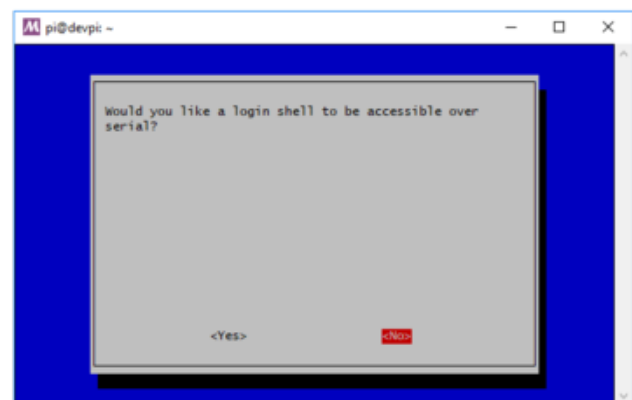
Run `sudo raspi-config` and select the following:



Interfacing Options



Serial

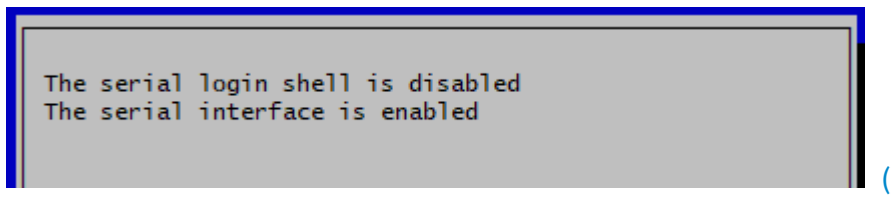


Select No on enabling the login shell



Select Yes on enabling serial port hardware

Once complete you should have no console and yes on serial interface:



Then reboot

Once you've rebooted, you can use the built in UART via `/dev/ttyS0` or `/dev/serial0`.

Reboot your Pi

After rebooting the Pi for the above changes to take effect, you can proceed with running gpsd

Restart GPSD with HW UART

Restart gpsd and redirect it to use HW UART instead of the USB port we pointed it to earlier. Simply entering the following two commands.

For the Raspberry Pi 1 or 2 (but NOT the 3!) run these commands:

```
$ sudo killall gpsd
$ sudo gpsd /dev/ttyAMA0 -F /var/run/gpsd.sock
```

And for the Raspberry Pi 3 or the Pi Zero W, run these commands to use the different serial port:

```
sudo killall gpsd  
sudo gpsd /dev/serial0 -F /var/run/gpsd.sock
```

As with the USB example, you can test the output with:

```
$ cgps -s
```

Further Resources

[Don't forget to also read our bigger tutorial on the Ultimate GPS which has lots more details, datasheets and examples for setting the sentences, update rate, etc! \(\)](#)

The following tutorials may be useful to you if you want to dig into this a bit further, and do something a bit more advanced with your GPS data:

- [GETTING GPS TO WORK ON A RASPBERRY PI \(\)](#).
- [GPSD Client How-To \(\)](#)
- [A nice writeup of using GPSd with python using threads to make it faster \(\)](#)

Doing something fun with GPS and tracking data? Be sure to post about it in the [Adafruit forums \(\)](#) so everyone else can get inspired by it!