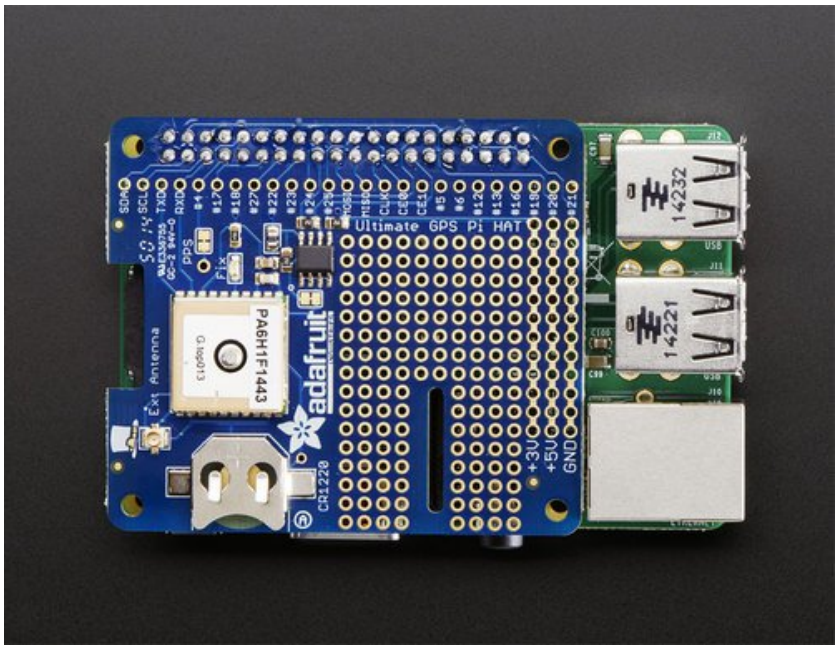
Adafruit Ultimate GPS HAT for Raspberry Pi

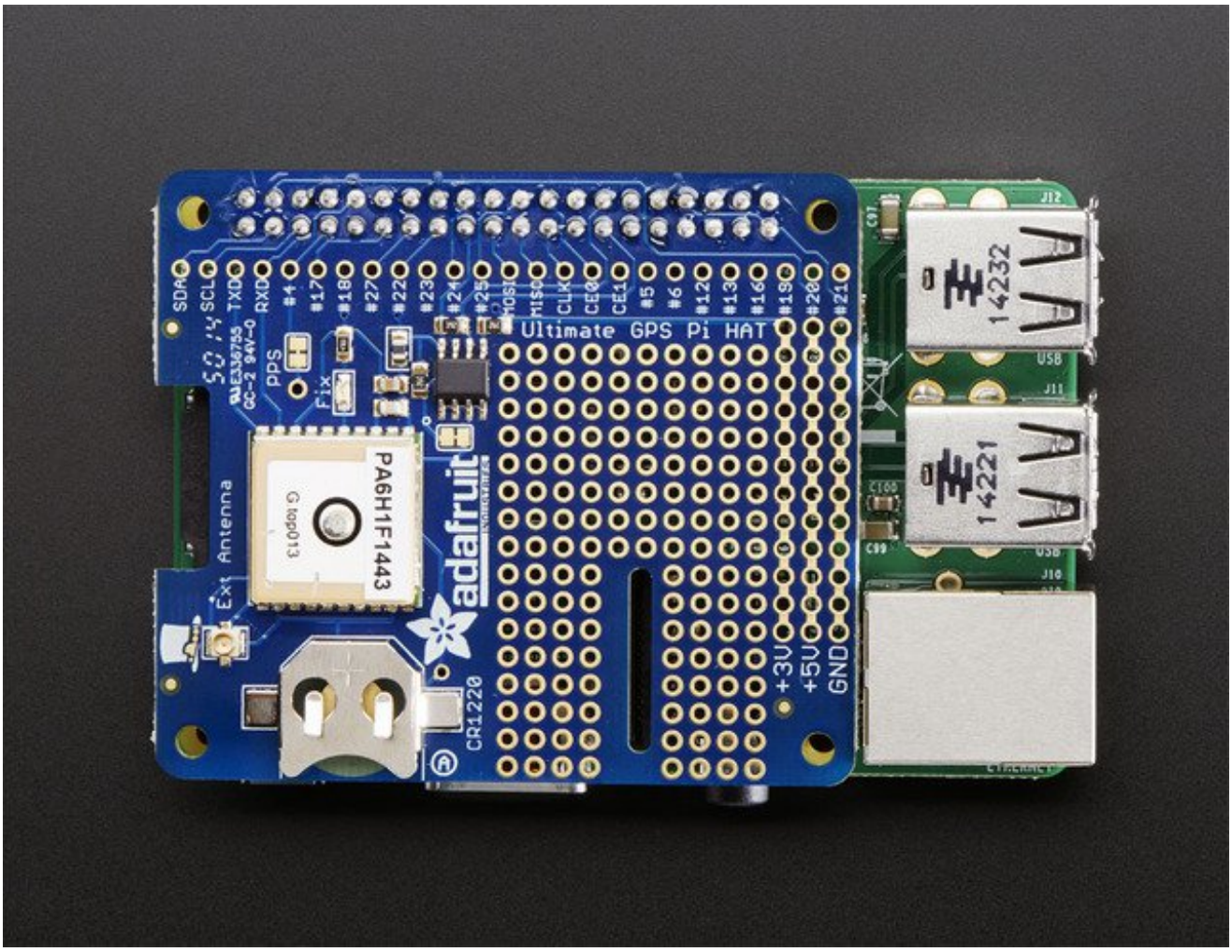
Created by lady ada



Last updated on 2016-10-06 07:07:32 PM UTC

Guide Contents

Overview



It's 10PM, do you know where your Pi is? If you had this Pi HAT, you would! This new HAT from Adafruit adds our celebrated Ultimate GPS on it, so you can add precision time and location to your Pi 1 Model A+ or B+, or Pi 2

This does not yet work on the Pi 3 due to intense firmware/hardware changes that affected the hardware UART. Please stick to Pi 2 or 'lower' for now!

Here's the low-down on the GPS module:

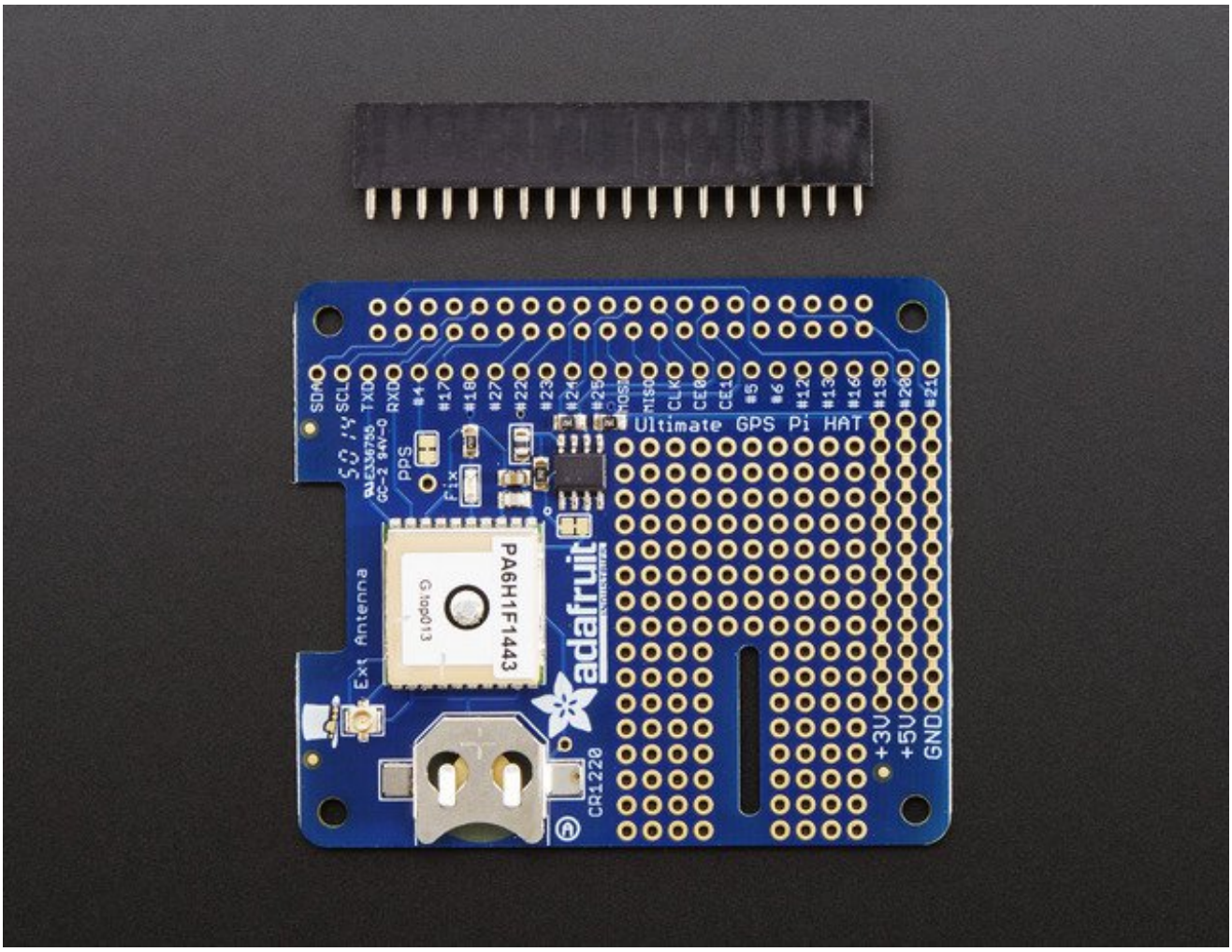
- -165 dBm sensitivity, 10 Hz updates, 66 channels
- Only 20mA current draw
- Built in Real Time Clock (RTC) - slot in a CR1220 backup battery for 7-years or more of timekeeping even if the Pi is off!

- PPS output on fix, by default connected to pin #4
- We have received reports that it works up to ~32Km altitude (the GPS theoretically does not have a limit until 40Km)
- Internal patch antenna which works quite well when used outdoors + u.FL connector for external active antenna for when used indoors or in locations without a clear sky view
- Fix status LED blinks to let you know when the GPS has determined the current coordinates

Please note, this HAT takes over the Pi's hardware UART to send/receive data to and from the GPS module. So, if you need to use the RX/TX pins with a console cable, you cannot also use this HAT. Instead, you'll have to use a composite or HDMI monitor and keyboard to log in, or use *ssh* to connect over the network to your Pi.

Comes as a fully assembled GPS + PCB and an additional 2x20 GPIO header. Some light soldering is required to attach the 2x20 GPIO header to the HAT but it's fast and easy for anyone with a soldering iron and solder. You can also swap the plain female header we have with a 'stacky' type that lets you plug in a hat or GPIO cable on top (http://adafru.it/2223) or a slim ultra-low-profile header. (http://adafru.it/eja)

Pinouts



This HAT is pretty simple but its worth going thru what pins are used for what!

The Ultimate GPS uses the following GPIO pins: **TXD RXD #4** and **EEDATA/EECLK**

Serial Console Pins

The Raspberry Pi has only one serial port, and you do need serial to chat to a GPS so we will take over the **RXD** and **TXD** pins.

PPS Pin

GPS's can output a 'pulse per second' for synchronizing the time. We have a breakout for this and a closed jumper that connects it to **GPIO#4** - if you do not need PPS and want to use #4 for something else, just cut the little connection on the PPS PCB line.

HAT EEPROM

EEDATA/EECLK are connected to a 24C32 EEPROM that contains the name and maker of this HAT (it's not used by the Pi Kernel just yet)

Pi GPS Setup

This does not yet work on the Pi 3 due to intense firmware/hardware changes that affected the hardware UART. Please stick to Pi 2 or 'lower' for now!

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! (http://adafru.it/aTl)

Once you follow these steps, you will no longer have a log in console on the Serial RX/TX pins, you will have to use a monitor+keyboard or ssh into your Pi! (or remove the HAT and undo the edits)

Set up the Pi to release the console pins

Start by logging into your Pi using a monitor/keyboard or via**ssh**

Step One: Edit /boot/cmdline.txt

If you are using **Raspbian Wheezy or Jessie**, enter the following command from the command line:

```
sudo nano /boot/cmdline.txt
```

And change:

dwc_otg.lpm_enable=0 console=ttyAMA0,115200 kgdboc=ttyAMA0,115200 console=tty1 root=/dev/mmcblk0p2 rootfstype=ext4 elevator=deadline rootwait

to:

dwc_otg.lpm_enable=0 console=tty1 root=/dev/mmcblk0p2 rootfstype=ext4 elevator=deadline rootwait

(eg, remove console=ttyAMA0,115200 and if there, kgdboc=ttyAMA0,115200)

Raspbian Wheezy only

Step Two: Edit /etc/inittab

On a Wheezy machine, from the command prompt enter the following command:

```
sudo nano /etc/inittab
```

And change:

```
#Spawn a getty on Raspberry Pi serial line
T0:23:respawn:/sbin/getty -L ttyAMA0 115200 vt100
```

to:

```
#Spawn a getty on Raspberry Pi serial line
#T0:23:respawn:/sbin/getty -L ttyAMA0 115200 vt100
```

That is, **add a #** to the beginning of the line!

Raspbian Jessie

Step Two:

Run the following two commands to stop and disable the tty service:

```
sudo systemctl stop serial-getty@ttyAMA0.service
sudo systemctl disable serial-getty@ttyAMA0.service
```

Step Three: Reboot your Pi

Run **sudo shutdown -h now** to shutdown your Pi safely.

Remove power, attach the GPS hat and re-power the Pi

After rebooting the Pi for the above changes to take effect, you can proceed with testing the GPS!

Basic Test

We can get started by testing the GPS right at the command line.

Before you start, you'll want to make sure the GPS modules has a 'fix' - that means it has received enough data from satellites to determinet he location.

You can quickly determine fix status by looking at the Red LED on the GPS Hat:

- Blinking on and off once every other second - NO FIX
- Blinking once per 10 seconds or so - HAS FIX

You may need to put the Pi near a window or attach an external antenna. Once it has a fix, continue!

Serial test

Lets stat by just reading the 'raw' data from the GPS on the command line. Run:

```
stty -F /dev/ttyAMA0 raw 9600 cs8 clocal -cstopb
```

To set **/dev/ttyAMA0** (the built in terminal console) to raw data (**raw**) 9600 baud (**9600**) 8-bits (**cs8**) no modem control (**-clocal**) and 1 stop bit (**cstopb**)

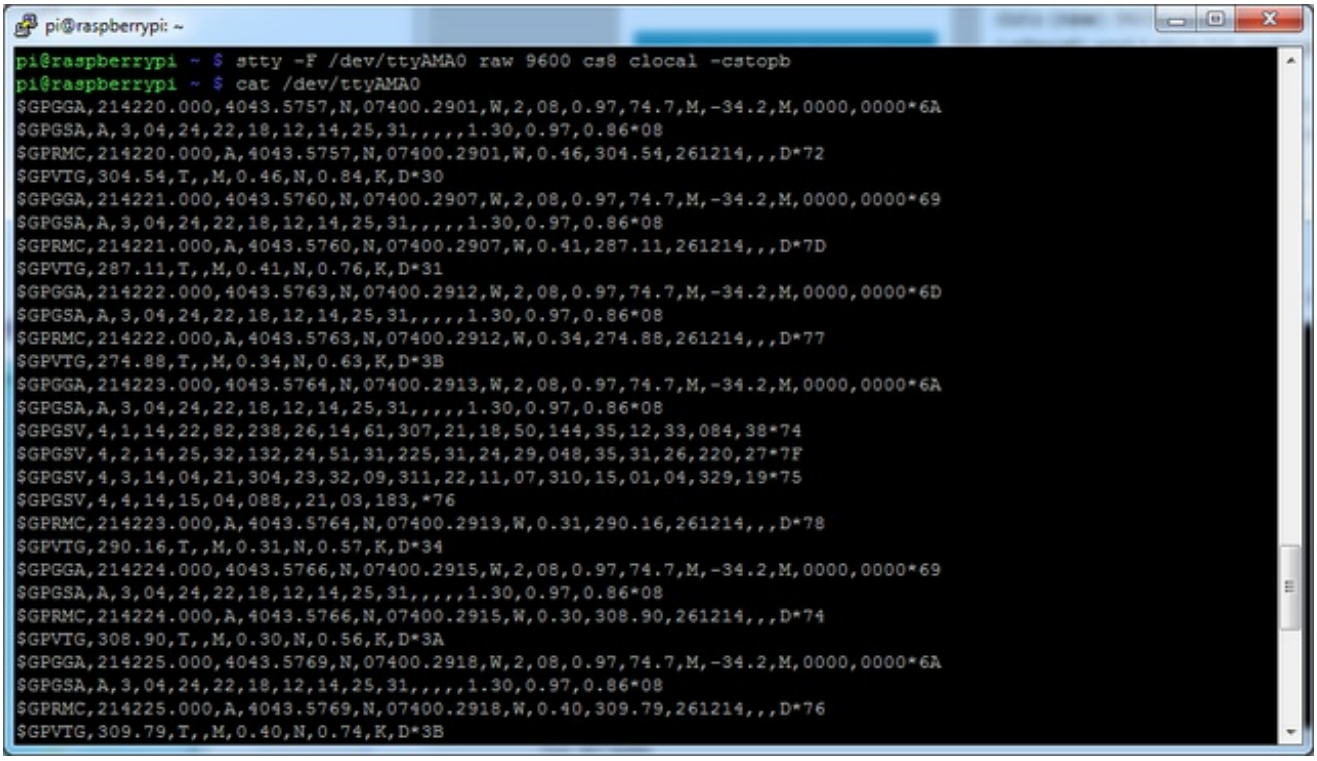
It's pretty picky so please use those settings and not noodle with them too much unless you're sure!

Now run

```
cat /dev/ttyAMA0
```

To read raw data directly from the GPS HAT

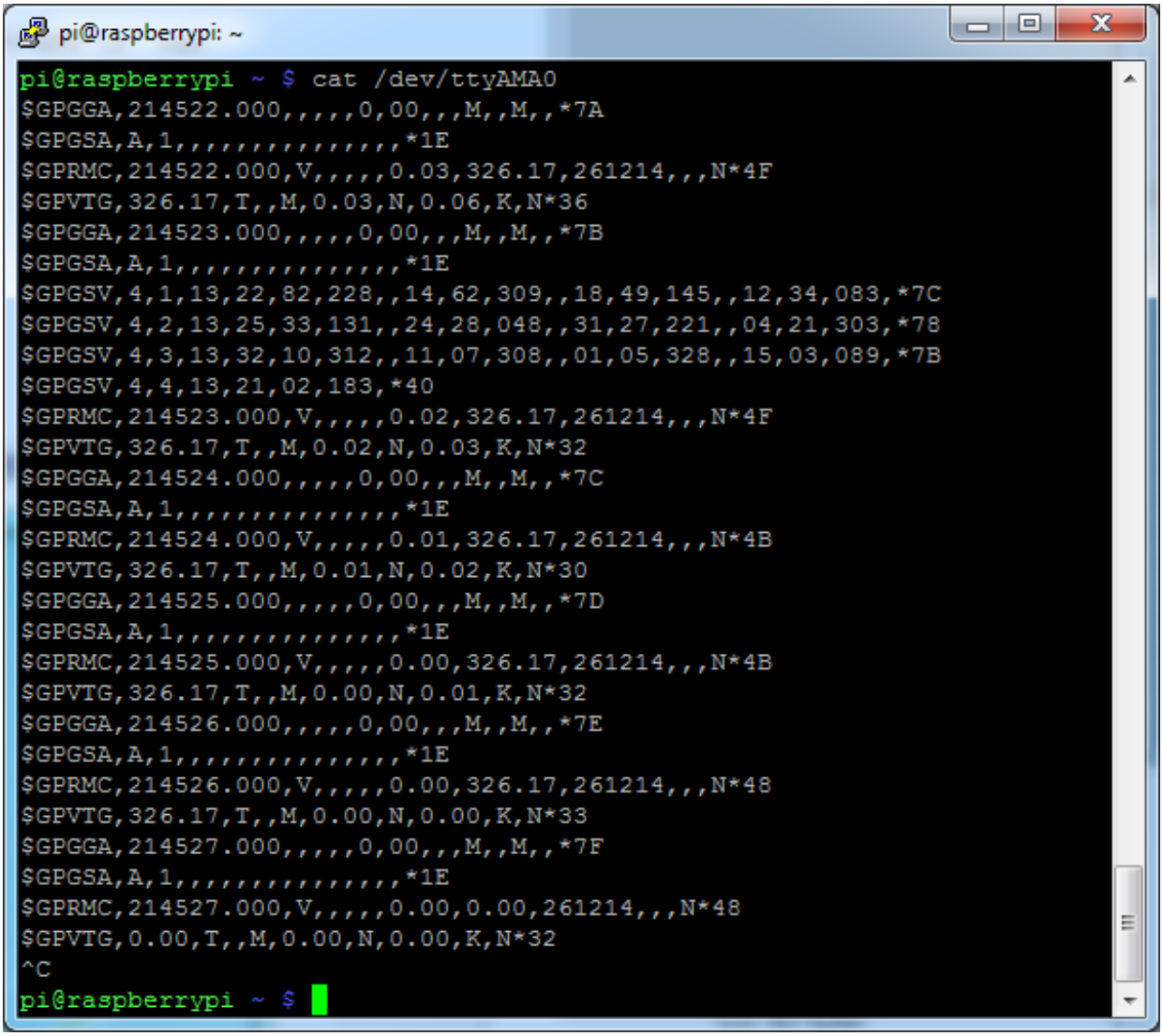
You should see something like this!



This is the raw GPS "NMEA sentence" output from the GPS module. There are a few different kinds of NMEA sentences, the most common ones people use are the **$GPRMC** (**G**lobal **P**ositioning **R**ecommended **M**inimum **C**oordinates or something like that) and the **$GPGGA** sentences. These two provide the time, date, latitude, longitude, altitude, estimated land speed, and fix type. Fix type indicates whether the GPS has locked onto the satellite data and received enough data to determine the location (2D fix) or location+altitude (3D fix).

For more details about NMEA sentences and what data they contain, check out this site (http://adafru.it/aMk)

If your data looks like this, with a lot of commas, with no data in between them. That's because this module is does not have a 'fix'. **DONT FORGET** To get a fix, we need to put the module or antenna outside or have it able to see the sky!



GPS modules will always send data EVEN IF THEY DO NOT HAVE A FIX! In order to get 'valid' (not-blank) data you must have the GPS module directly outside, with the square ceramic antenna pointing up with a clear sky view. In ideal conditions, the module can get a fix in under 45 seconds. however depending on your location, satellite configuration, solar flares, tall buildings nearby, RF noise, etc it may take up to half an hour (or more) to get a fix! This does not mean your GPS module is broken, the GPS module will always work as fast as it can to get a fix.

OK lets look at a good NMEA sentence with 'fix'

Something that starts with $GPRMC like:

$GPRMC,194509.000,A,4042.6142,N,07400.4168,W,2.03,221.11,160412,,,A*77

This line is called the RMC (Recommended Minimum) sentence and has pretty much all of the most useful data. Each chunk of data is separated by a comma.

The first part **194509.000** is the current time **GMT** (Greenwich Mean Time). The first two numbers **19** indicate the hour (1900h, otherwise known as 7pm) the next two are the minute, the next two are the seconds and finally the milliseconds. So the time when this screenshot was taken is 7:45 pm and 9 seconds. The GPS does not know what time zone you are in, or about "daylight savings" so you will have to do the calculation to turn GMT into your timezone

The second part is the 'status code', if it is a **V** that means the data is **V**oid (invalid). If it is an **A** that means its **A**ctive (the GPS could get a lock/fix)

The next 4 pieces of data are the geolocation data. According to the GPS, my location is **4042.6142,N** (Latitude 40 degrees, 42.6142 decimal minutes North) &**07400.4168,W**. (Longitude 74 degrees, 0.4168 decimal minutes West) To look at this location in Google maps, type **+40° 42.6142', -74° 00.4168'** into the google maps search box (http://adafru.it/aMl) . Unfortunately gmaps requires you to use +/- instead of NSWE notation. N and E are positive, S and W are negative.

People often get confused because the GPS is working but is "5 miles off" - this is because they are not parsing the lat/long data correctly. Despite appearances, the geolocation data is NOT in decimal degrees. It is in degrees and minutes in the following format: Latitude: DDMM.MMMM (The first two characters are the degrees.) Longitude: DDDMM.MMMM (The first three characters are the degrees.)

The next data is the ground speed in knots. We're going **2.03** knots

After that is the tracking angle, this is meant to approximate what 'compass' direction we're heading at based on our past travel

The one after that is **160412** which is the current date (this sentence was first 'grabbed' on 16th of April, 2012).

Finally there is the ***XX** data which is used as a data transfer checksum

Once you get a fix using your GPS module, verify your location with google maps (or some other mapping software). Remember that GPS is often only accurate to 5-10 meters and worse if you're indoors or surrounded by tall buildings.

Use 'gpsd'

You can always just read that raw data, but its much nicer if you can have some Linux software prettify it. We'll try out **gpsd** which is a GPS-handling Daemon (background-helper)

Installing a GPS Daemon (gpsd)

The first step is installing some software on your Raspberry Pi that understands the serial data that your GPS module is providing via /dev/ttyAMA0.
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, make sure your Pi has an Internet connection and run the following commands from the console:

```
sudo apt-get update
sudo apt-get install gpsd gpsd-clients python-gps
```

And install the software as it prompts you to do.

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
```

Try out 'gpsd'

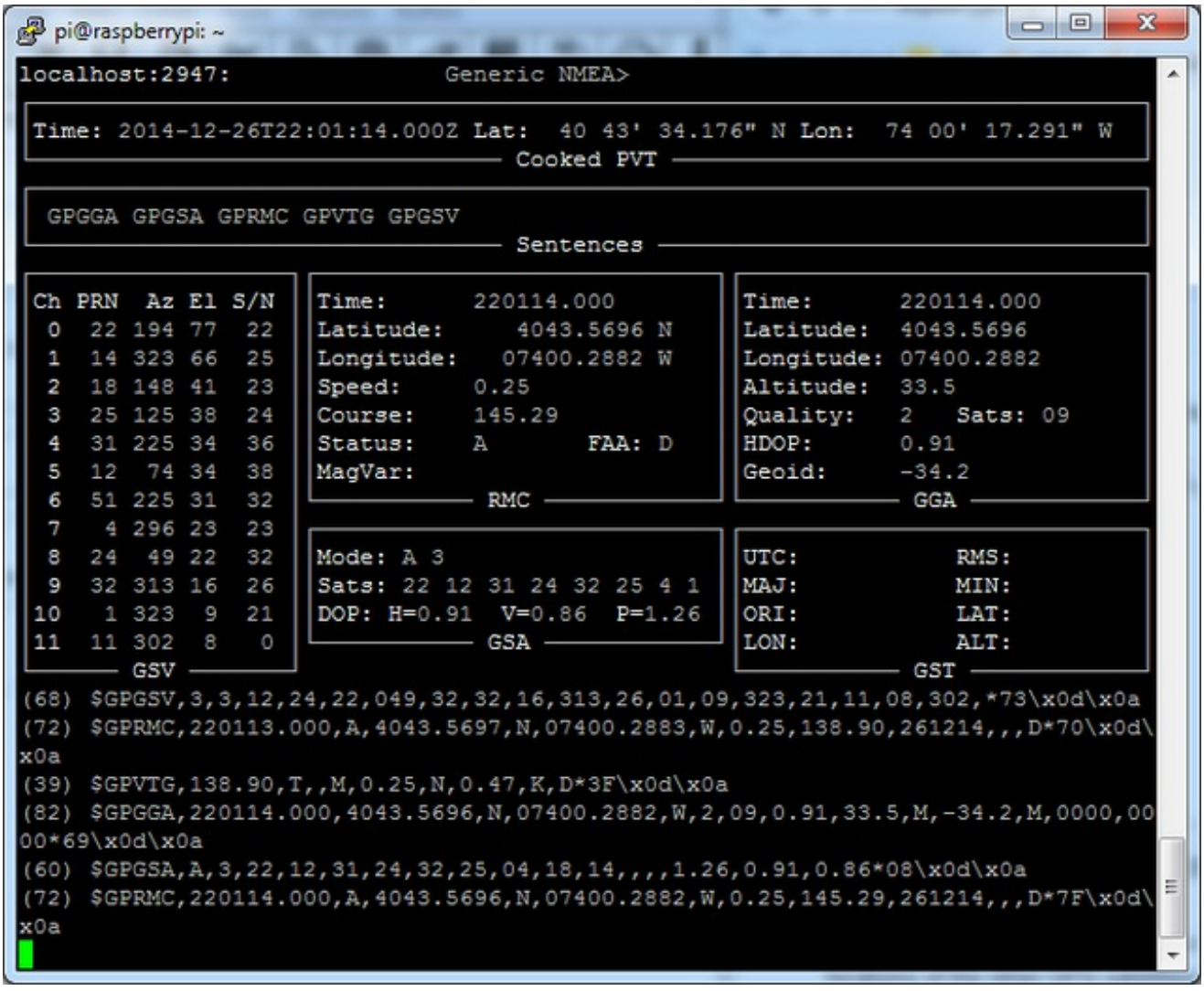
After installing gpsd and disabling the gpsd systemd service as mentioned above you're ready to start using gpsd yourself.

Start **gpsd** and direct it to use HW UART. Simply entering the following command:

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

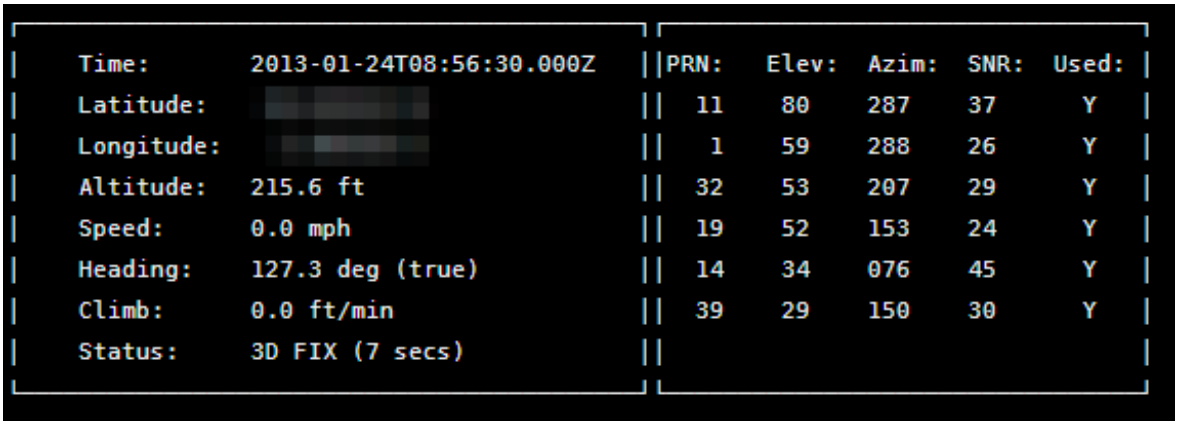
... which will point the gps daemon to our GPS device on the**/dev/ttyAMA0** console

Try running **gpsmon** to get a live-streaming update of GPS data!



or **cgps** which gives a less detailed, but still quite nice output

```
cgps -s
```



Don't forget, you do need to have FIX to use these tools! If you have FIX 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/ttyAMA0 -F /var/run/gpsd.sock
```

More info!

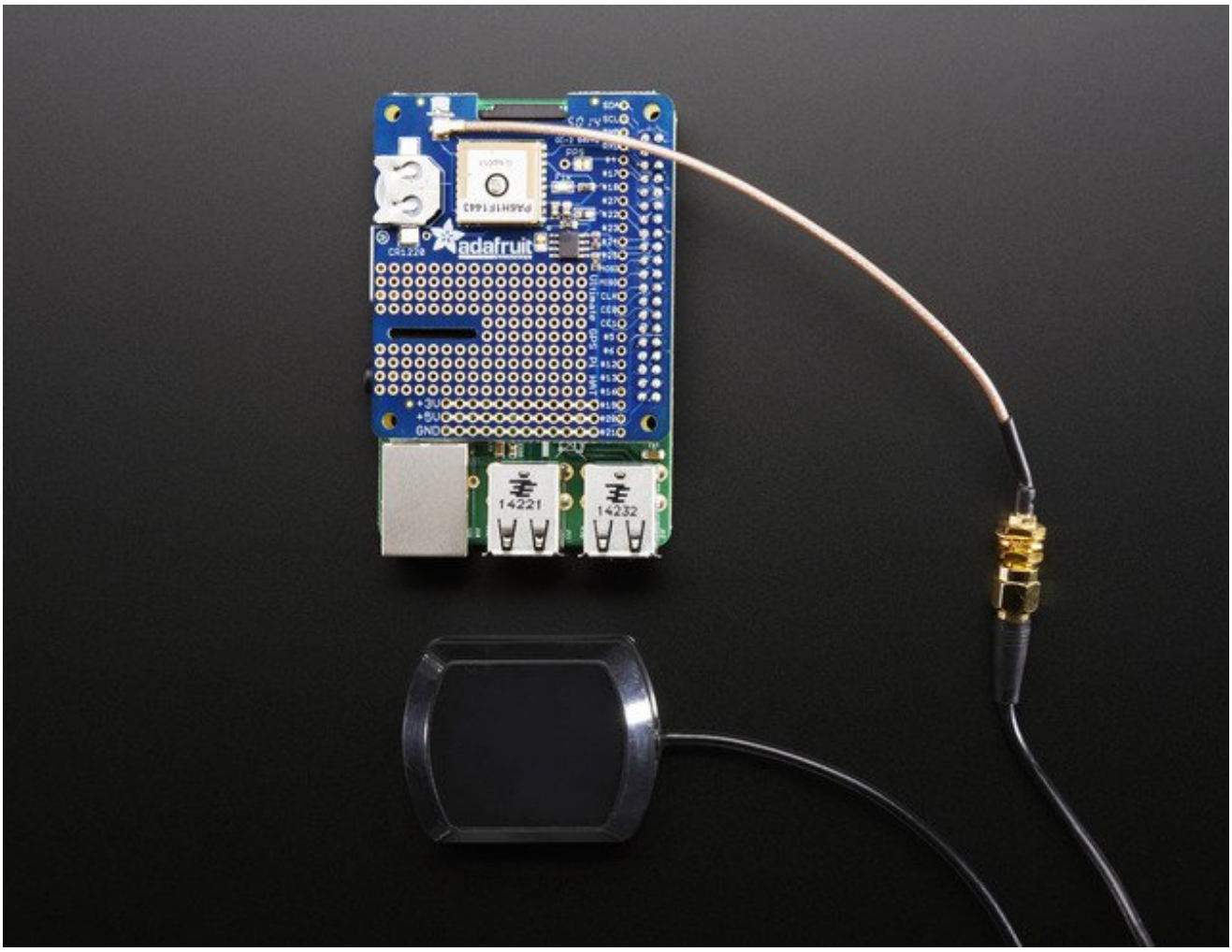
You can use **gpsd** to help you read data into other programming languages like C and Python using a variety of existins libraries

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 (http://adafru.it/aWL).
- GPSD Client How-To (http://adafru.it/eUC) ... including examples in C, C++ and Python
- The official GPSD project pages (http://adafru.it/aWN)
- A nice writeup of using GPSd with python using threads to make it faster (http://adafru.it/cGM)
- Great forum post on how to setup time synchronization with a Raspberry Pi and the PPS output of the GPS HAT. (http://adafru.it/fEw)
- Another good write-up on how to configure a Pi to use GPS and the PPS output for time synchronization. (http://adafru.it/fPy) Note that this guide uses GPIO18 for the PPS signal but the HAT uses GPIO4.

Doing something fun with GPS and tracking data? Be sure to post about it in the Adafruit forums (http://adafru.it/forums) so everyone else can get inspired by it!

External Antenna



All Ultimate GPS modules have a built in patch antenna - this antenna provides -165 dBm sensitivity and is perfect for many projects. However, if you want to place your project in a box, it might not be possible to have the antenna pointing up, or it might be in a metal shield, or you may need more sensitivity. In these cases, you may want to use an external active antenna. (http://adafru.it/960)

Active antennas draw current, so they do provide more gain but at a power cost. Check the antenna datasheet for exactly how much current they draw - its usually around 10-20mA. (http://adafru.it/960)

Most GPS antennas use SMA connectors, which are popular and easy to use. However, an SMA connector would be fairly big on the GPS breakout so we went with a uFL connector -

which is lightweight, small and easy to manufacture. If you don't need an external antenna it wont add significant weight or space but its easy to attach a uFL->SMA adapter cable (http://adafru.it/851). Then connect the GPS antenna to the cable.

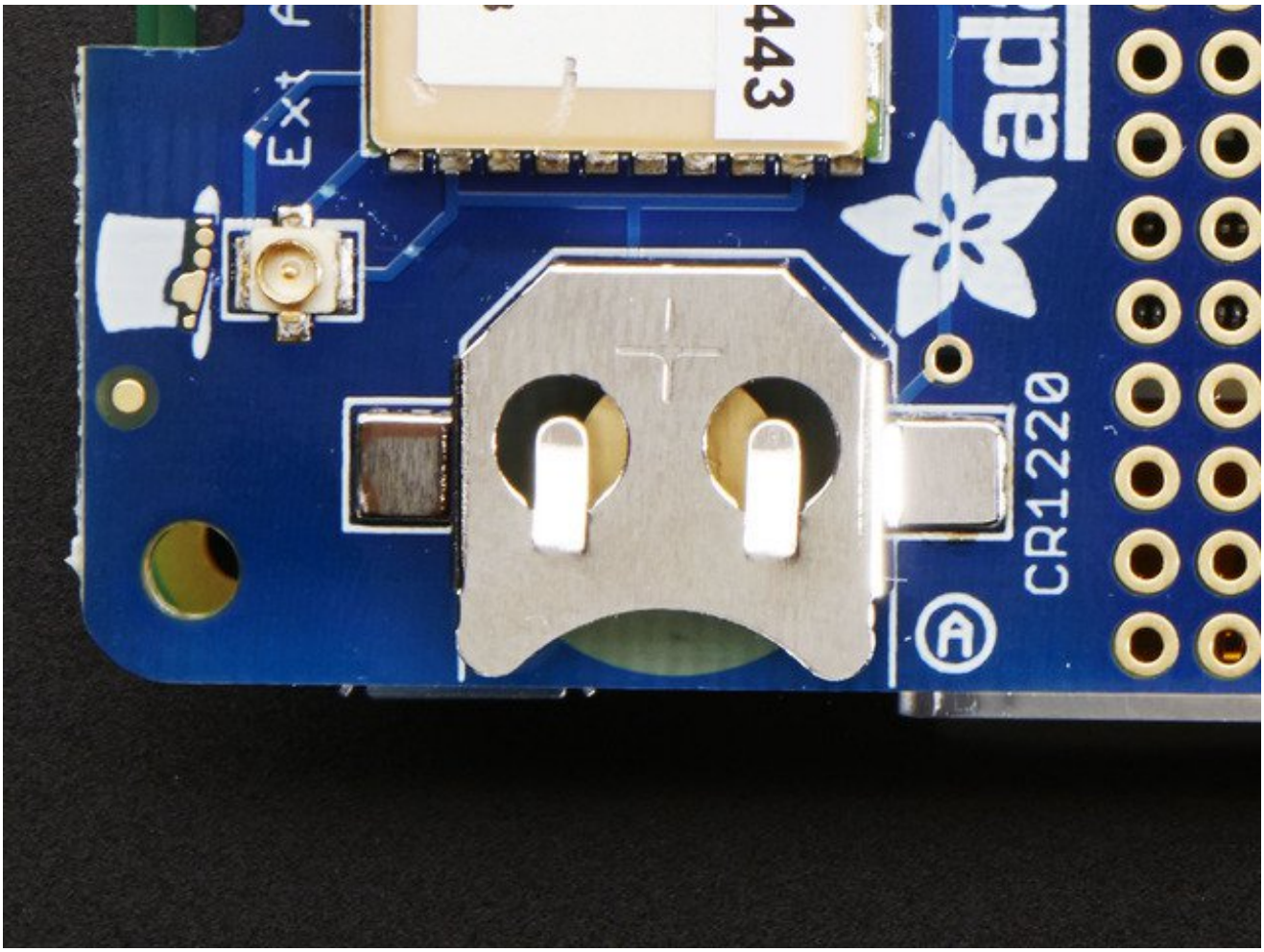
The Ultimate GPS will automagically detect an external active antenna is attached and 'switch over' - you do not need to send any commands

There is an output sentence that will tell you the status of the antenna.**$PGTOP,11,x** where **x** is the status number. If **x** is **3** that means it is using the external antenna. If**x** is **2** it's using the internal antenna and if **x** is **1** there was an antenna short or problem.

On newer shields & modules, you'll need to tell the firmware you want to have this report output, you can do that by sending it the command **$PGCMD,33,1*6C**

Battery Backup

The GPS has a built in real time clock, which can keep track of time even when it power is lost and it doesn't have a fix yet. It can also help reduce fix times, if you expect to have a flakey power connection (say you're using solar or similar). To use the RTC, we need to insert a battery. There is holder on the HAT for a **CR1220** sized battery holder. We provide the holder but the battery is not included. You can use any 12mm coin cell - these are popular and we also carry them in the Adafruit shop.



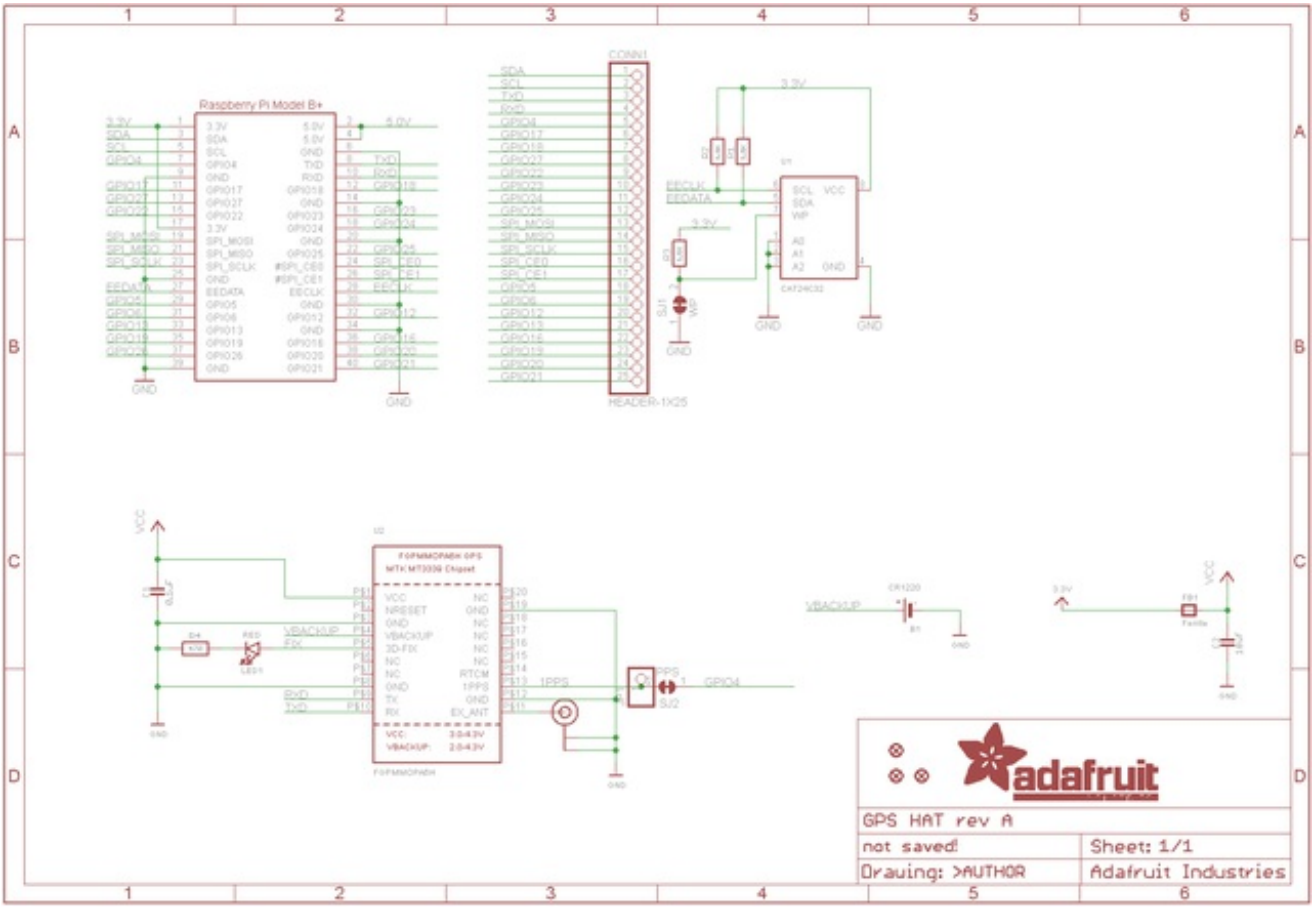
Remember, the GPS does not know what time zone you are in (even though it knows your location, there is no easy way to determine time zone without a massive lookup table) so all date/time data is in UTC (aka. Greenwich Mean Time)

Downloads

Files

- MTK3329/MTK3339 command set sheet (http://adafru.it/e7A) for changing the fix data rate, baud rate, sentence outputs, etc!
- PMTK 'complete' datasheet (http://adafru.it/d2Q) (like the above but with even more commands)
- Datasheet for the PA6H (MTK3339) GPS module itself (http://adafru.it/aPO)
- Fritzing object in the Adafruit Fritzing Library (http://adafru.it/aP3)
- EagleCAD PCB files on GitHub (http://adafru.it/rKE)

Schematic



Fabrication Print

Fabrication Print

Dimensions in inches