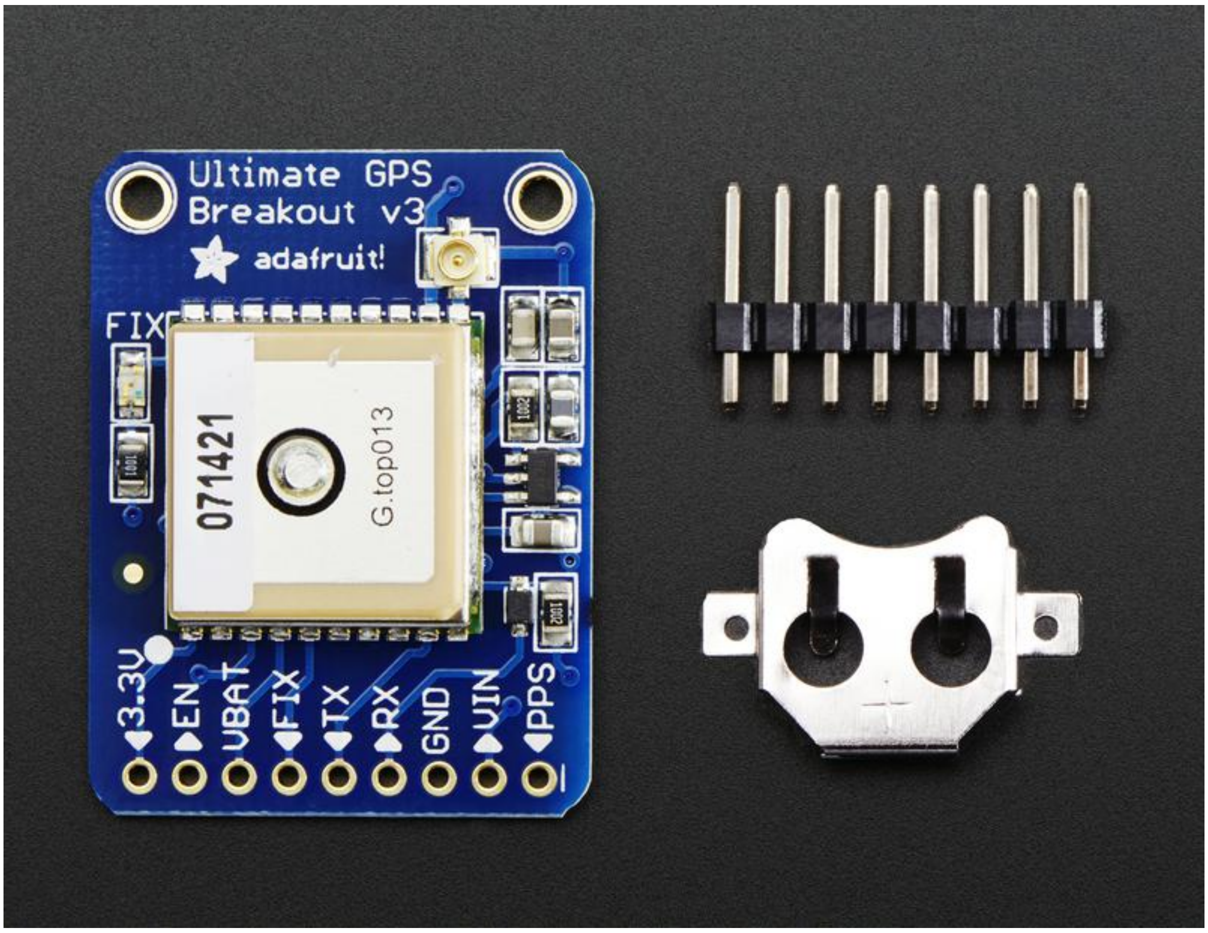
Adafruit Ultimate GPS

Created by lady ada

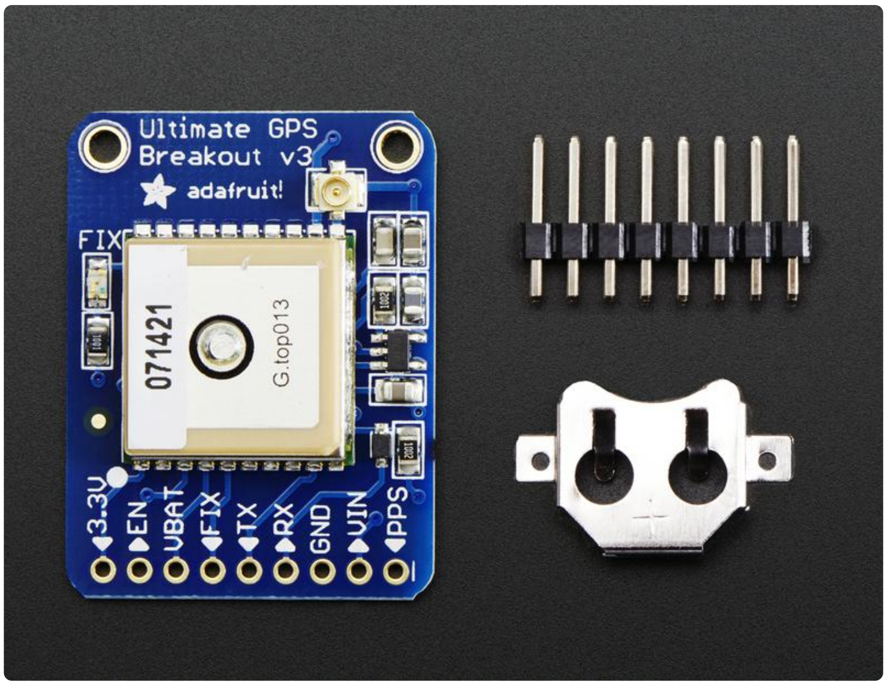


https://learn.adafruit.com/adafruit-ultimate-gps

Last updated on 2022-12-01 01:51:00 PM EST

Table of Contents

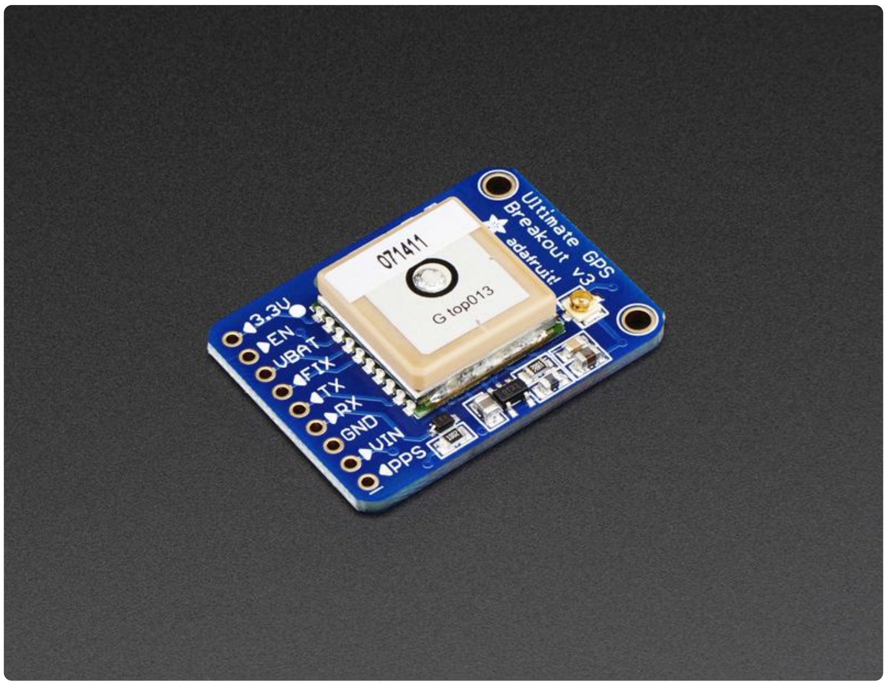
Overview



We carry a few different GPS modules here in the Adafruit shop, but none that satisfied our every desire - that's why we designed this little GPS breakout board. We believe this is the Ultimate GPS module, so we named it that. It's got everything you want and more:

- -165 dBm sensitivity, 10 Hz updates, 66 channels
- 5V friendly design and only 20mA current draw
- Breadboard friendly + two mounting holes
- RTC battery-compatible
- Built-in datalogging
- PPS output on fix
- Internal patch antenna + u.FL connector for external active antenna
- Fix status LED

The breakout is built around the MTK3339 chipset, a no-nonsense, high-quality GPS module that can track up to 22 satellites on 66 channels, has an excellent high-sensitivity receiver (-165 dB tracking!), and a built in antenna. It can do up to 10 location updates a second for high speed, high sensitivity logging or tracking. Power usage is incredibly low, only 20 mA during navigation.

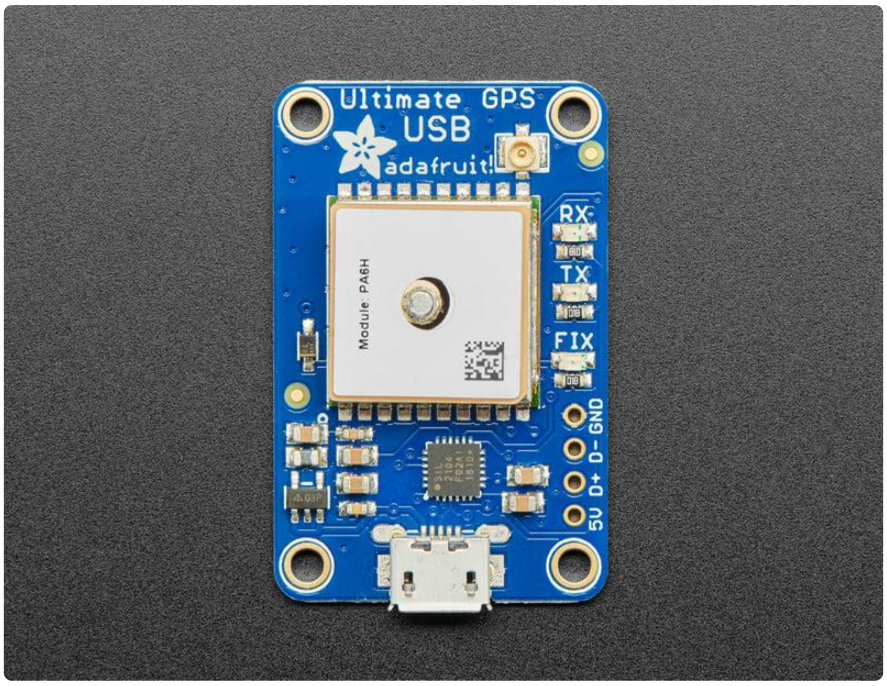


Ultimate GPS Breakout Board

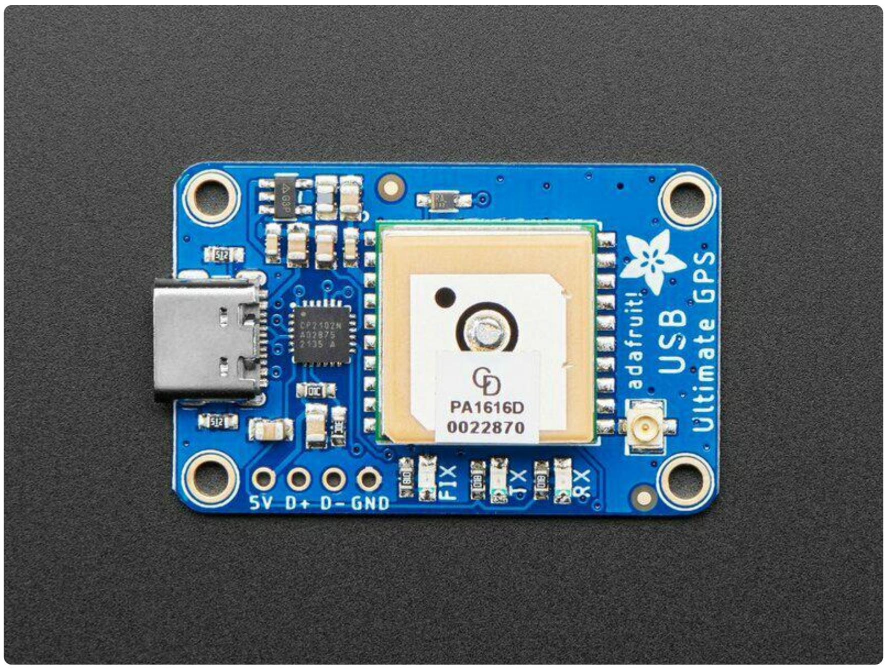
The Breadboard Breakout board comes with: a ultra-low dropout 3.3V regulator so you can power it with 3.3-5VDC in, 5V level safe inputs, ENABLE pin so you can turn off the module using any microcontroller pin or switch, a footprint for optional CR1220 coin cell to keep the RTC running and allow warm starts and a tiny bright red LED. The LED blinks at about 1Hz while it's searching for satellites and blinks once every 15 seconds when a fix is found to conserve power. If you want to have an LED on all the time, we also provide the FIX signal out on a pin so you can put an external LED on.

Ultimate GPS USB Board

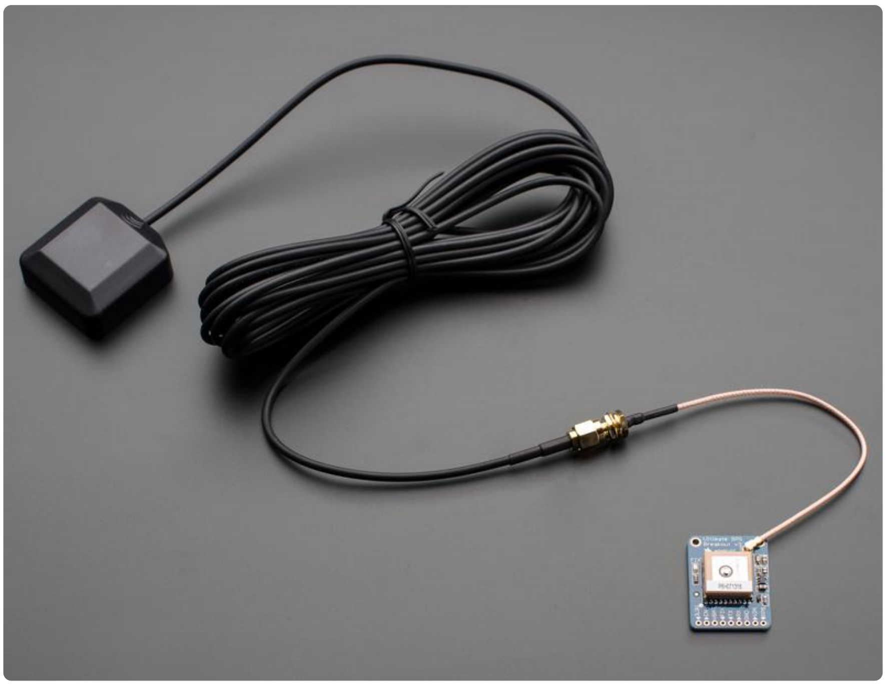
The USB Breakout board comes with: 4-pin USB breakout for direct-soldering or connection to a USB host, two yellow receive/transmit LEDs let you know when data is being transmitted to or from the GPS module serial interface, a footprint for optional CR1220 coin cell to keep the RTC running and allow warm starts and a tiny bright red LED. The LED blinks at about 1Hz while it's searching for satellites and blinks once every 15 seconds when a fix is found to conserve power. If you want to have an LED on all the time, we also provide the FIX signal out on a pin so you can put an external LED on.



As of March 11, 2022 - we have given this board a makeover! It now comes with a modern USB C connector on the board (instead of micro USB), and a CP2102N instead of CP2104 (functionally-compatible but newer)



Antenna Usage



Two features that really stand out about version 3 MTK3339-based module is the external antenna functionality and the the built in data-logging capability. The module has a standard ceramic patch antenna that gives it -165 dB sensitivity, but when you want to have a bigger antenna, you can snap on any 3V active GPS antenna via the uFL connector. The module will automatically detect the active antenna and switch over! Most GPS antennas use SMA connectors so you may want to pick up one of our uFL to SMA adapters. (http://adafru.it/851)

The other cool feature of the new MTK3339-based module (which we have tested with great success) is the built in datalogging ability. Since there is a microcontroller inside the module, with some empty FLASH memory, the newest firmware now allows sending commands to do internal logging to that FLASH. The only thing is that you do need to have a microcontroller send the "Start Logging" command. However, after that message is sent, the microcontroller can go to sleep and does not need to wake up to talk to the GPS anymore to reduce power consumption. The time, date, longitude, latitude, and height is logged every 15 seconds and only when there is a fix. The internal FLASH can store about 16 hours of data, it will automatically append data so you don't have to worry about accidentally losing data if power is lost. It is not possible to change what is logged and how often, as its hardcoded into the module but we found that this arrangement covers many of the most common GPS datalogging requirements.

Pick one up today at the Adafruit shop! (http://adafru.it/746)

Specifications:

Module specs:

- Satellites: 22 tracking, 66 searching
- Patch Antenna Size: 15mm x 15mm x 4mm
- Update rate: 1 to 10 Hz
- Position Accuracy: 1.8 meters
- Velocity Accuracy: 0.1 meters/s
- Warm/cold start: 34 seconds
- Acquisition sensitivity: -145 dBm
- Tracking sensitivity: -165 dBm
- Maximum Velocity: 515m/s
- Vin range: 3.0-5.5VDC
- MTK3339 Operating current: 25mA tracking, 20 mA current draw during navigation
- Output: NMEA 0183, 9600 baud default
- DGPS/WAAS/EGNOS supported
- FCC E911 compliance and AGPS support (Offline mode : EPO valid up to 14 days)
- Up to 210 PRN channels
- Jammer detection and reduction
- Multi-path detection and compensation

Breakout board details:

- Weight (not including coin cell or holder): 8.5g
- Dimensions (not including coin cell or holder): 25.5mm x 35mm x 6.5mm / 1.0" x 1.35" x 0.25"

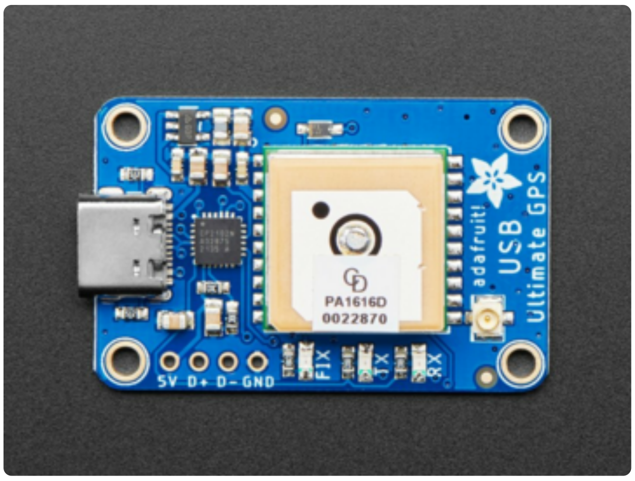
If you purchased a module before March 26th, 2012 and it says MTK3329 on the silkscreen, you have the PA6B version of this breakout with the MT3329 chipset. The MTK3329 does not have built in datalogging. If your module has sharpie marker crossing out the MTK3329 text or there is no text, you have a PA6C MTK3339 with datalogging ability. If you have the version with "v3" next to the name, you have the PA6H which has PPS output and external-antenna support

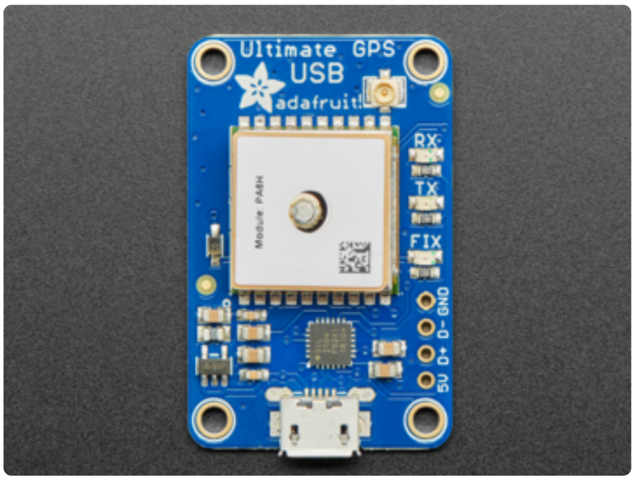
This tutorial assumes you have a '3339 type module.

Pinouts

adafruit_products_gps_746_topkit.jpg

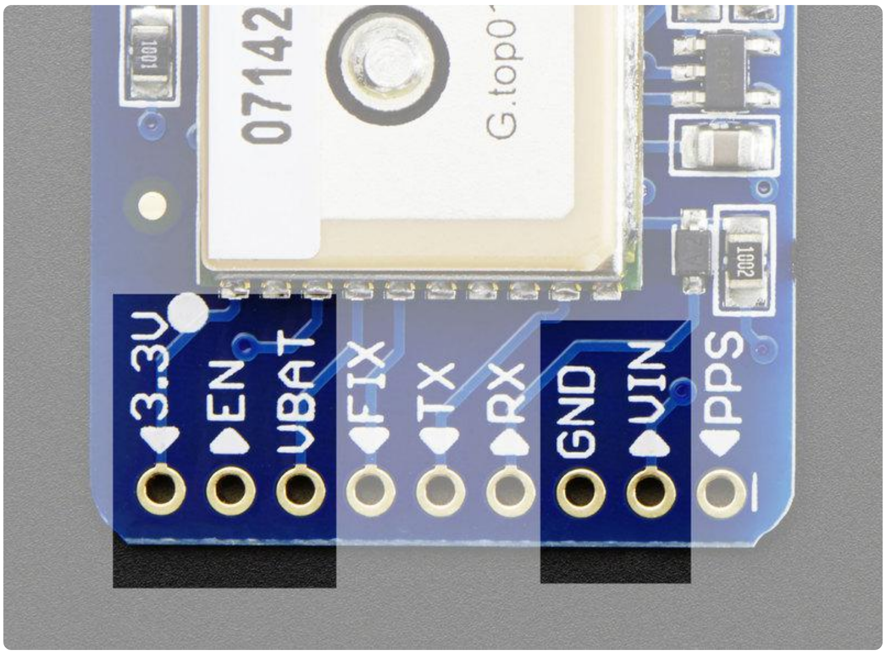
The plain GPS breakout is meant to be used with a microcontroller that has a 3/5V UART and has all the GPIO pins exposed on the bottom 0.1" header





The USB version doesn't have those GPIO pins broken out. Instead it has a single USB serial port. There's 4 pads to the right side that are the USB pinouts for when you want to solder it directly to a USB host controller.

Breakout Power Pins



These are the pins that are involved with powering the GPS. Starting from the right are the required power pins:

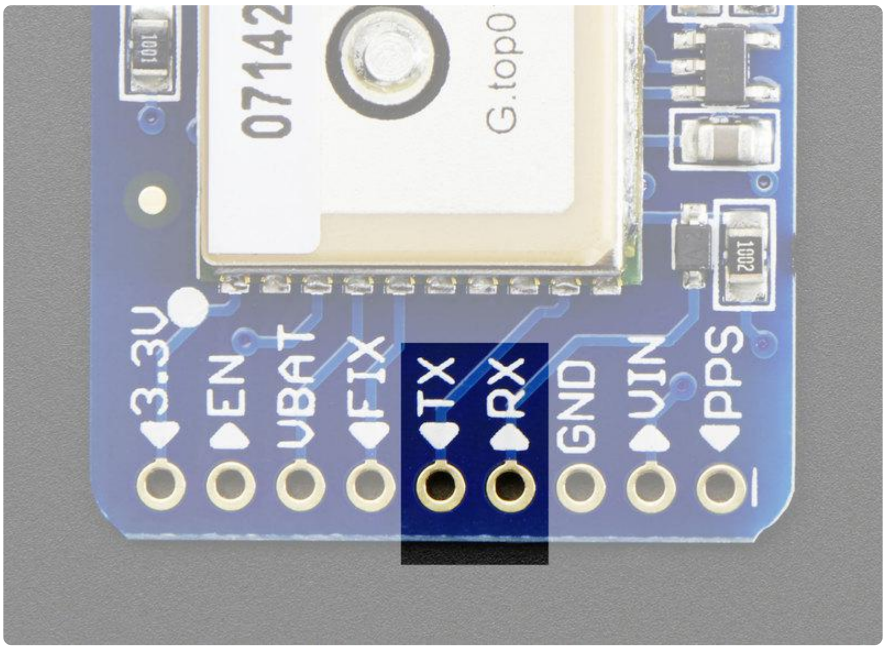
- VIN - power input, connect to 3-5VDC. It's important to connect to a clean and quiet power supply. GPS's are very sensitive, so you want a nice and quiet power supply. Don't connect to a switching supply if you can avoid it, an LDO will be less noisy!
- GND - power and signal ground. Connect to your power supply and microcontroller ground.

Then, on the left are some optional power pins:

- VBAT is an input pin - it is connected to the GPS real time clock battery backup. We suggest using the battery spot on the back but if you have a project with a coin cell or other kind of battery that you want to use (and its under 3.3V) you can connect it to the VBAT pin. For V1 and V2 modules: If you do this, be sure to cut the trace on the back between the RTC solder pads
- EN is the Enable pin, it is pulled high with a 10K resistor. When this pin is pulled to ground, it will turn off the GPS module. This can be handy for very low power projects where you want to easily turn the module off for long periods. You will lose your fix if you disable the GPS and it will also take a long time to get fix back if you dont have the backup battery installed.

- 3.3V is the output from the onboard 3.3V regulator. If you have a need for a clean 3.3V output, you can use this! It can provide at least 100mA output.

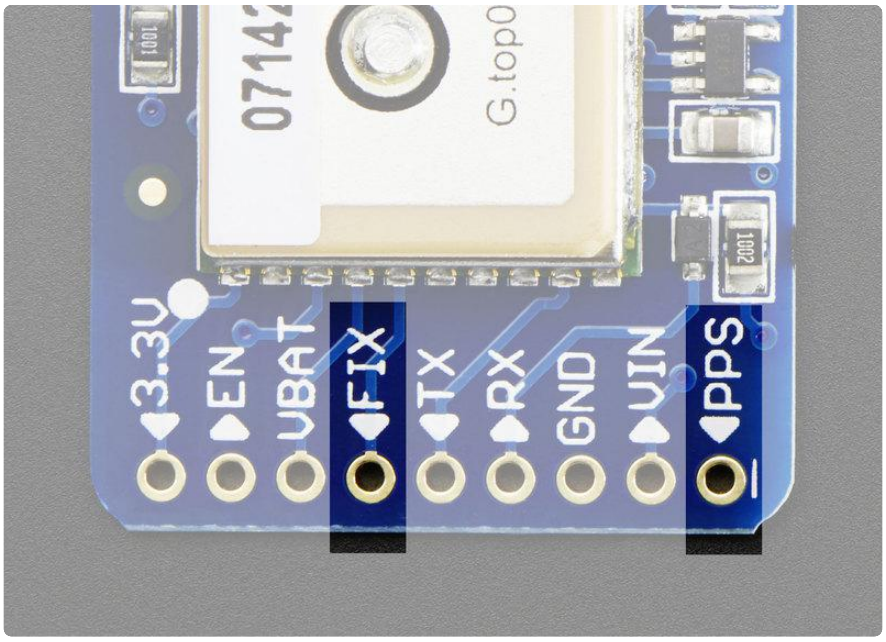
Breakout Serial Data Pins



Next pins you'll want to use are the serial data pins:

- TX - the pin that transmits data from the GPS module to your microcontroller or computer. It is 3.3V logic level. Data comes out at 9600 baud by default
- RX - the pin that you can use to send data to the GPS. You can use use 3.3V or 5V logic, there is a logic level shifter. By default it expects 9600 baud data, and remember you need to send it checksum'med NMEA sentences

Breakout Other Pins



FIX is an output pin - it is the same pin as the one that drives the red LED. When there is no fix, the FIX pin is going to pulse up and down once every second. When there is a fix, the pin is low (0V) for most of the time, once every 15 seconds it will pulse high for 200 milliseconds

PPS is a new pin output on V3 modules. Its a "pulse per second" output. Most of the time it is at logic low (ground) and then it pulses high (3.3V) once a second, for 50-100ms, so it should be easy for a microcontroller to sync up to it

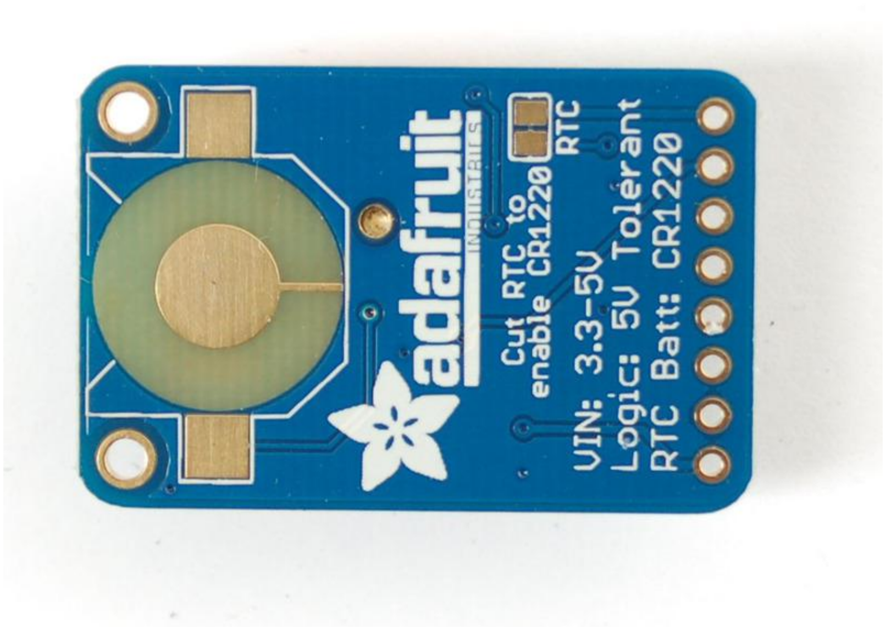
---

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 attach a battery. There is a spot on the back 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.

Normally, if the GPS loses power, it will revert to the factory default for baud rates, configuration, etc. A backup battery will mean that those defaults will not be lost!

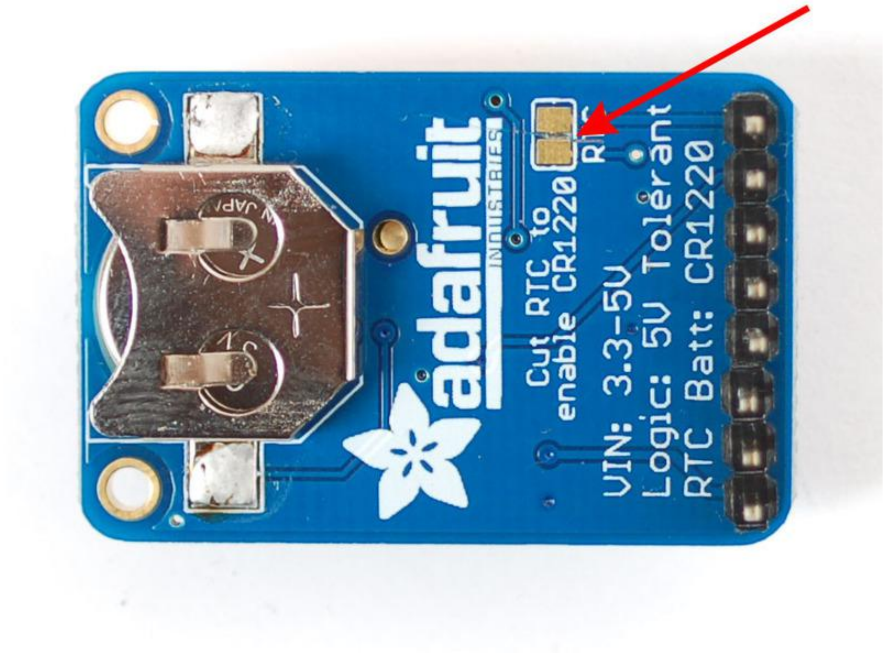
The backup real-time-clock circuitry draws 7 uA (0.007 mA) so a CR1220 will last 40mAh / 0.007mA = 5,714 hours = 240 days continuously. The backup battery is only used when there's no main 3V power to the GPS, so as long as it's only used as backup once in a while, it will last years



If you have a v1 or v2 module ONLY:

Before inserting a battery into the battery holder, first cut the trace between the two solder pads on the back, labeled RTC (this disconnects the VIN pin from the battery input) Use a multimeter with continuity checking to verify the two pads are no longer tied together.

V3 modules do not have a trace to cut, they have a built-in diode!

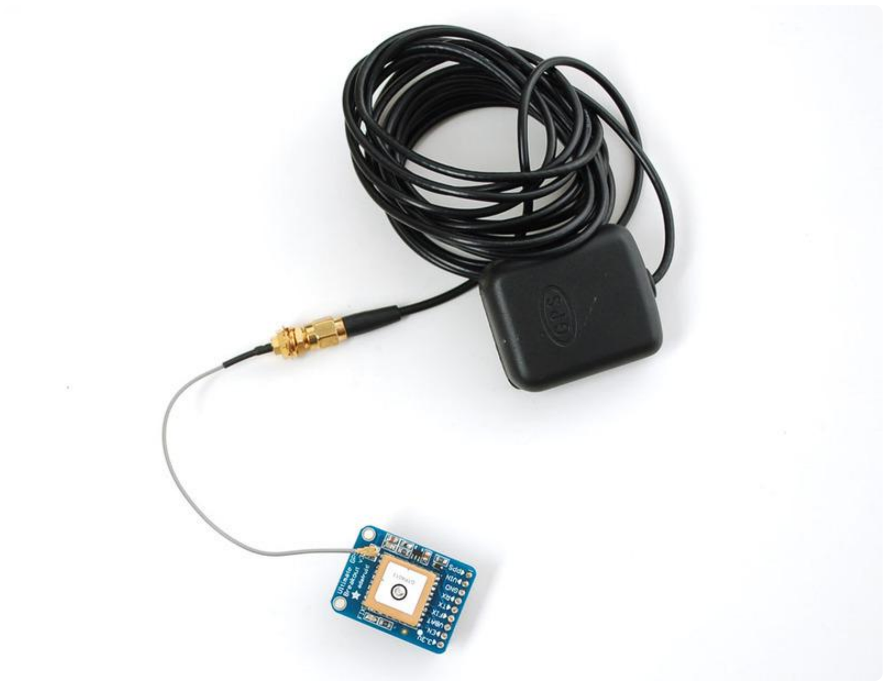


Remember, the GPS does not know what time zone you are in (even though it knows your location, there is no easy way to determine time zone without a massive lookup table) so all date/time data is in UTC (aka. Greenwich Mean Time) - You will have to write the code that converts that to your local time zone and account for Daylight Savings if required! Since that's pretty complicated, most people just stick to keeping everything in UTC

External Antenna

New in version 3 of the Ultimate GPS breakout, we now have support for optional external antennas!

This is not available in v1 or v2 so if you do not see the uFL connector, you have an older version of the module which cannot support antennas



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.

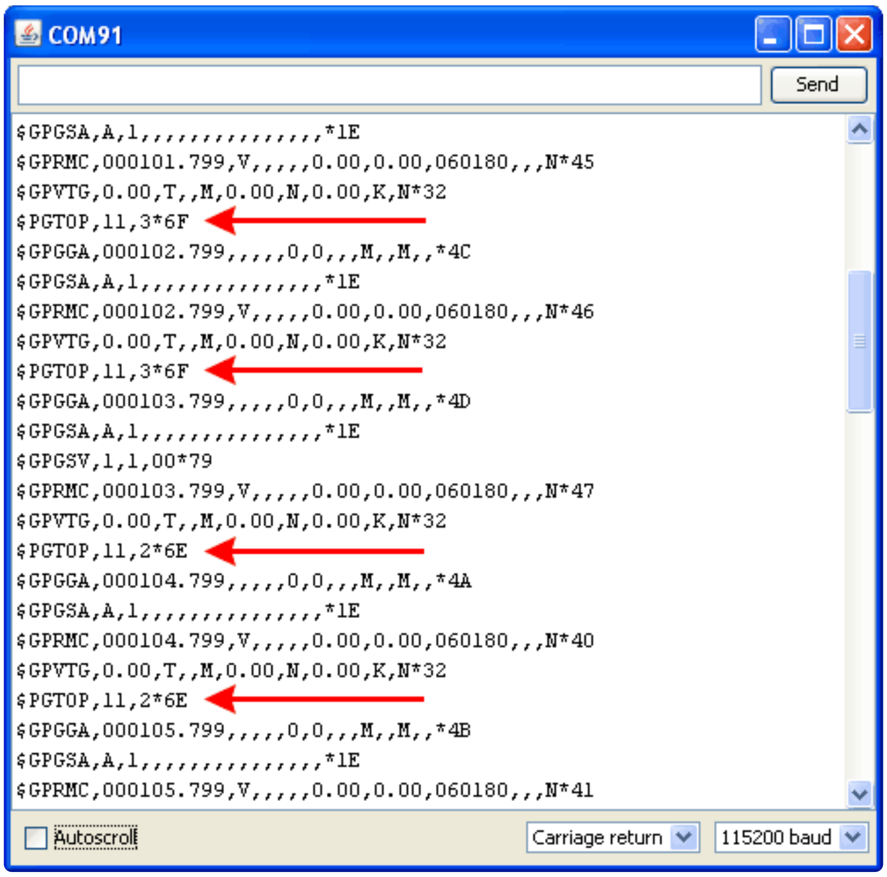
However, an SMA connector would be fairly big on the GPS breakout so we went with a uFL connector - which is lightweight, small and easy to manufacture. If you don't need an external antenna it wont add significant weight or space but its easy to attach a uFL->SMA adapter cable (http://adafru.it/851). Then connect the GPS antenna to the cable.

uFL connectors are small, delicate and are not rated for strain or tons of connections/disconnections. Once you attach a uFL adapter use strain relief to avoid ripping off the uFL

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 i s 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 adding a gps.sendCommand(PGCMD_ANTENNA) ar ound the same time you set the update rate/sentence output.



```
$GPGSA,A,1,,,,,,,,,,,,,,,*1E
$GPRMC,000101.799,V,,,,,0.00,0.00,060180,,,N*45
$GPVTG,0.00,T,,M,0.00,N,0.00,K,N*32
$PGTOP,11,3*6F
$GPGGA,000102.799,,,,,0,0,,,M,,M,,*4C
$GPGSA,A,1,,,,,,,,,,,,,,,*1E
$GPRMC,000102.799,V,,,,,0.00,0.00,060180,,,N*46
$GPVTG,0.00,T,,M,0.00,N,0.00,K,N*32
$PGTOP,11,3*6F
$GPGGA,000103.799,,,,,0,0,,,M,,M,,*4D
$GPGSA,A,1,,,,,,,,,,,,,,,*1E
$GPGSV,1,1,00*79
$GPRMC,000103.799,V,,,,,0.00,0.00,060180,,,N*47
$GPVTG,0.00,T,,M,0.00,N,0.00,K,N*32
$PGTOP,11,2*6E
$GPGGA,000104.799,,,,,0,0,,,M,,M,,*4A
$GPGSA,A,1,,,,,,,,,,,,,,,*1E
$GPRMC,000104.799,V,,,,,0.00,0.00,060180,,,N*40
$GPVTG,0.00,T,,M,0.00,N,0.00,K,N*32
$PGTOP,11,2*6E
$GPGGA,000105.799,,,,,0,0,,,M,,M,,*4B
$GPGSA,A,1,,,,,,,,,,,,,,,*1E
$GPRMC,000105.799,V,,,,,0.00,0.00,060180,,,N*41
```

Direct Computer Wiring

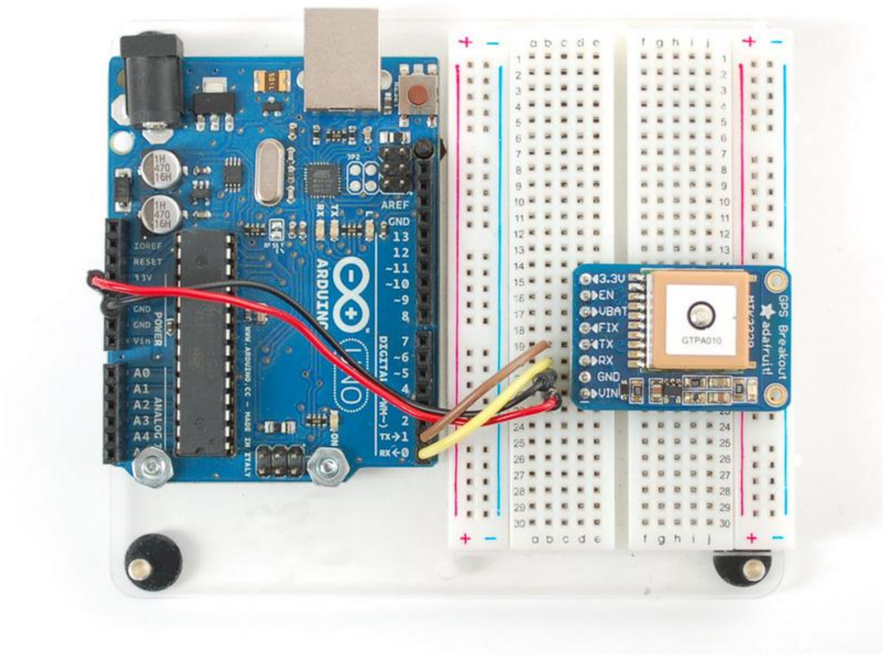
GPS modules are great in that the moment you turn them on, they'll start spitting out data, and trying to get a 'fix' (location verification). Like pretty much every GPS in existence, the Adafruit Ultimate GPS uses TTL serial output to send data so the best way to first test the GPS is to wire it directly to the computer via the TTL serial to USB converter on an Arduino. You can also use an FTDI Friend or other TTL adapter but for this demonstration we'll use a classic Arduino.

This tutorial step won't work with a chip that has NATIVE USB - its for an Arduino UNO compatible only. Go on to the next step, "Arduino Wiring", but refer back here for this discussion of the GPS data!

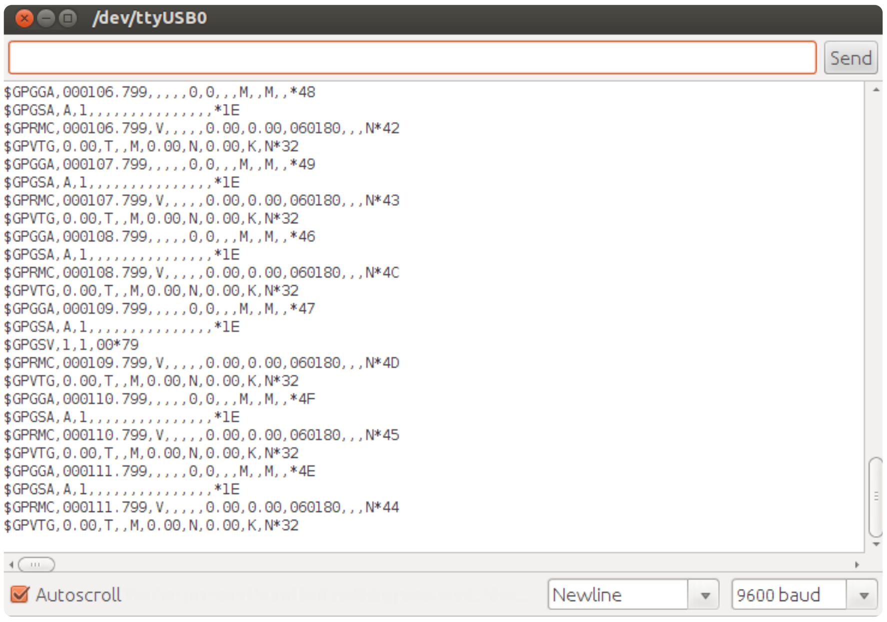
First, load a 'blank' sketch into the Arduino:

```
// this sketch will allow you to bypass the Atmega chip
// and connect the Ultimate GPS directly to the USB/Serial
// chip converter.

// Connect VIN to +5V
// Connect GND to Ground
// Connect GPS RX (data into GPS) to Digital 0
// Connect GPS TX (data out from GPS) to Digital 1

void setup() {}
void loop() {}
```

This is will free up the converter so you can directly wire and bypass the Arduino chip. Once you've uploaded this sketch, wire the GPS as follows. Your module may look slightly different, but as long as you are connecting to the right pin names, they all work identically for this part



Now plug in the USB cable, and open up the serial monitor from the Arduino IDE and be sure to select 9600 baud in the drop down. You should see text like the following:



```
$GPGGA,000106.799,,,,,0,0,,,M,,M,,*48
$GPGSA,A,1,,,,,,,,,,,,,,,*1E
$GPRMC,000106.799,V,,,,,0.00,0.00,060180,,,N*42
$GPVTG,0.00,T,,M,0.00,N,0.00,K,N*32
$GPGGA,000107.799,,,,,0,0,,,M,,M,,*49
$GPGSA,A,1,,,,,,,,,,,,,,,*1E
$GPRMC,000107.799,V,,,,,0.00,0.00,060180,,,N*43
$GPVTG,0.00,T,,M,0.00,N,0.00,K,N*32
$GPGGA,000108.799,,,,,0,0,,,M,,M,,*46
$GPGSA,A,1,,,,,,,,,,,,,,,*1E
$GPRMC,000108.799,V,,,,,0.00,0.00,060180,,,N*4C
$GPVTG,0.00,T,,M,0.00,N,0.00,K,N*32
$GPGGA,000109.799,,,,,0,0,,,M,,M,,*47
$GPGSA,A,1,,,,,,,,,,,,,,,*1E
$GPGSV,1,1,00*79
$GPRMC,000109.799,V,,,,,0.00,0.00,060180,,,N*4D
$GPVTG,0.00,T,,M,0.00,N,0.00,K,N*32
$GPGGA,000110.799,,,,,0,0,,,M,,M,,*4F
$GPGSA,A,1,,,,,,,,,,,,,,,*1E
$GPRMC,000110.799,V,,,,,0.00,0.00,060180,,,N*45
$GPVTG,0.00,T,,M,0.00,N,0.00,K,N*32
$GPGGA,000111.799,,,,,0,0,,,M,,M,,*4E
$GPGSA,A,1,,,,,,,,,,,,,,,*1E
$GPRMC,000111.799,V,,,,,0.00,0.00,060180,,,N*44
$GPVTG,0.00,T,,M,0.00,N,0.00,K,N*32
```

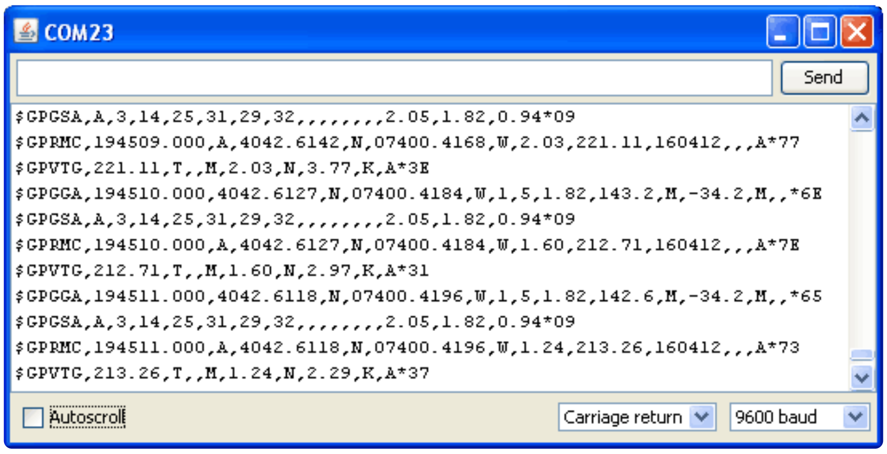
This is the raw GPS "NMEA sentence" output from the module. There are a few different kinds of NMEA sentences, the most common ones people use are the $GPRMC (Global Positioning RecommendedMinimum Coordinates or something like that) and the $GPGGA sentences. These two provide the time, date, latitude, 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 ()]()

If you look at the data in the above window, you can see that there are a lot of commas, with no data in between them. That's because this module is on my desk, indoors, and does not have a 'fix'. To get a fix, we need to put the module outside.

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.

If you can get a really long USB cord (or attach a GPS antenna to the v3 modules) and stick the GPS out a window, so its pointing at the sky, eventually the GPS will get a fix and the window data will change over to transmit valid data like this:



```
$GPGSA,A,3,14,25,31,29,32,,,,,,,,2.05,1.82,0.94*09
$GPRMC,194509.000,A,4042.6142,N,07400.4168,W,2.03,221.11,160412,,,A*77
$GPVTG,221.11,T,,M,2.03,N,3.77,K,A*3E
$GPGGA,194510.000,4042.6127,N,07400.4184,W,1,5,1.82,143.2,M,-34.2,M,,*6E
$GPGSA,A,3,14,25,31,29,32,,,,,,,,2.05,1.82,0.94*09
$GPRMC,194510.000,A,4042.6127,N,07400.4184,W,1.60,212.71,160412,,,A*7E
$GPVTG,212.71,T,,M,1.60,N,2.97,K,A*31
$GPGGA,194511.000,4042.6118,N,07400.4196,W,1,5,1.82,142.6,M,-34.2,M,,*65
$GPGSA,A,3,14,25,31,29,32,,,,,,,,2.05,1.82,0.94*09
$GPRMC,194511.000,A,4042.6118,N,07400.4196,W,1.24,213.26,160412,,,A*73
$GPVTG,213.26,T,,M,1.24,N,2.29,K,A*37
```

Look for the line that says $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 Void (invalid). If it is an A that means its Active (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 () . 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 (16th of April, 2012).

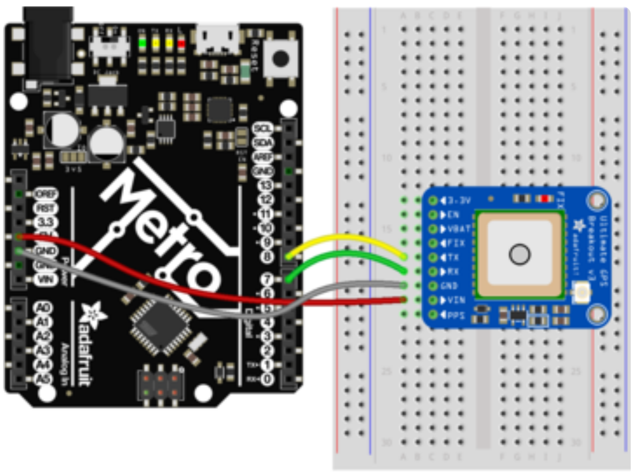
Finally there is the *XX data which is used as a data transfer checksum

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.

Breakout Arduino Wiring

Once you've gotten the GPS module tested with direct wiring, we can go forward and wire it up to a microcontroller. We'll be using an Arduino but you can adapt our code to any other microcontroller that can receive TTL serial at 9600 baud.

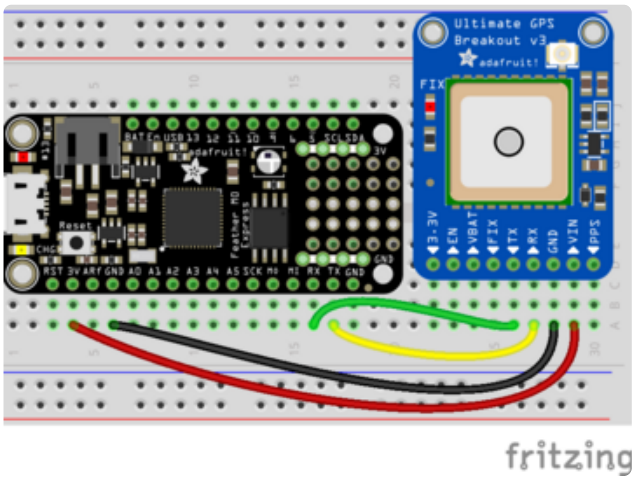
Software Serial Boards



For an Arduino compatible, or any board that does not have an extra `HardwareSerial` port, we'll use `SoftwareSerial`. The examples in the Adafruit_GPS Arduino library use pin 8 for Serial RX and pin 7 for Serial TX on the Arduino-compatible board. The constructor call in the example sketches is `SoftwareSerial(8, 7);`.

Connect VIN to +5V (or whatever the logic level is of your board), GND to Ground, RX to digital 7 and TX to digital 8. Note that the Arduino-compatible TX is connected to the GPS RX, and the Arduino-compatible RX is connected to the GPS TX.

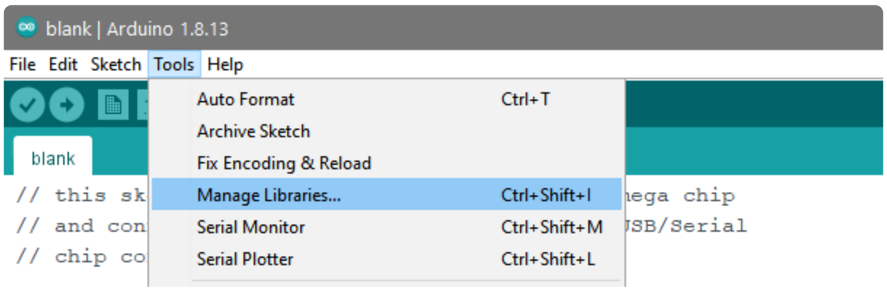
Hardware Serial Boards



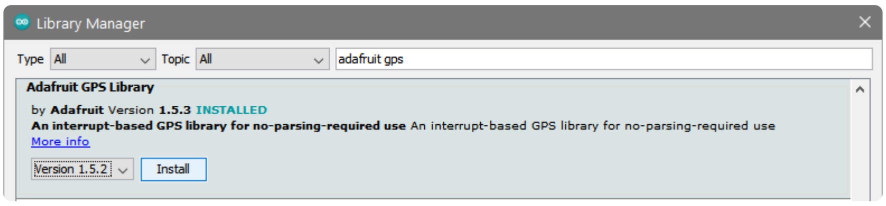
If you're using a board with hardware serial support, like this Feather M0, wire up VIN to 3.3V (since that is the logic level of the Feather M0), GND to ground, and GPS RX to Feather TX and GPS TX to Feather RX.

Next up, download the Adafruit GPS library. This library does a lot of the 'heavy lifting' required for receiving data from GPS modules, such as reading the streaming data in a background interrupt and auto-magically parsing it. You can check the code by visiting the GitHub repository ()

To install, open the Arduino library manager


blank | Arduino 1.8.13
File Edit Sketch Tools Help
blank
Auto Format Ctrl+T
Archive Sketch
Fix Encoding & Reload
Manage Libraries... Ctrl+Shift+I
Serial Monitor Ctrl+Shift+M
Serial Plotter Ctrl+Shift+L

Search for Adafruit GPS and click Install

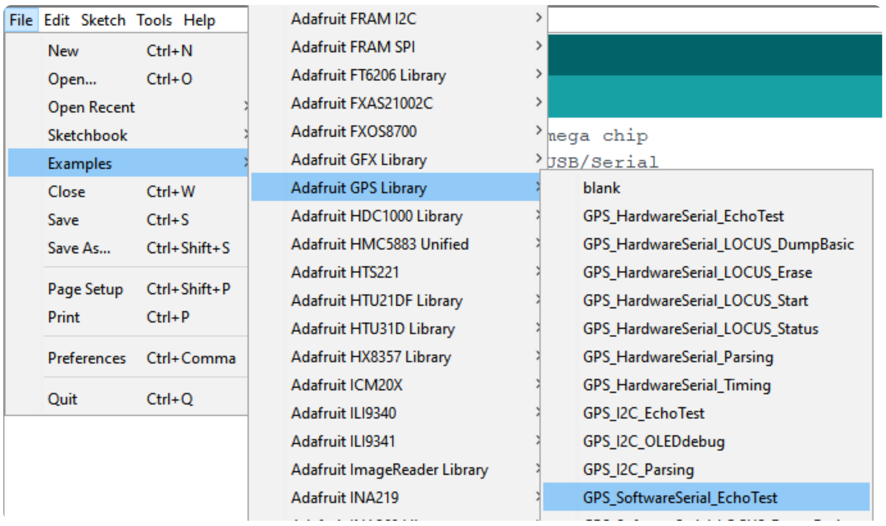


Adafruit GPS Library
by **Adafruit** Version **1.5.3** **INSTALLED**
An interrupt-based GPS library for no-parsing-required use An interrupt-based GPS library for no-parsing-required use

Library installation is a frequent stumbling block…if you need assistance, our All About Arduino Libraries () guide spells it out in detail!

Arduino UNO or other SoftwareSerial boards

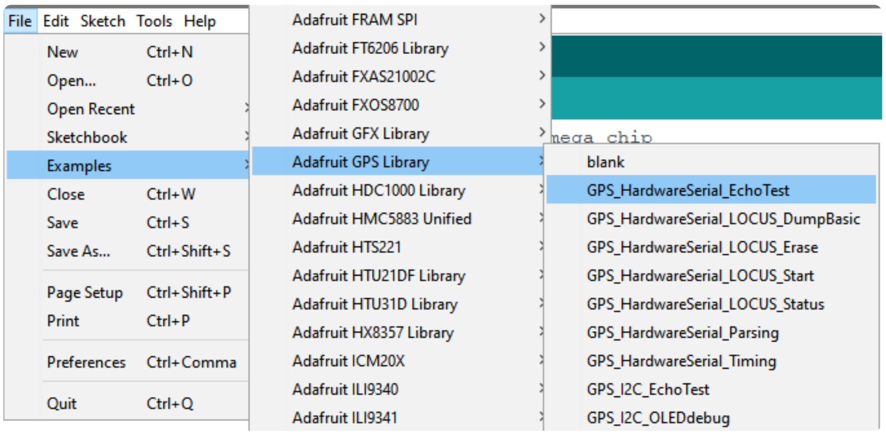
If your microcontrollers doesn't have a Hardware Serial interface available - say you're using an Arduino UNO or compatible, load up the `SoftwareSerial_echotest` example:



Leonardo/M0/M4/ESP32 and other Hardware Serial Boards

If you have a board with a spare Hardware Serial interface (which many boards other than the original Arduino UNO compatibles have) - we recommend using that since hardware serial will always give less errors and better performance than software serial!

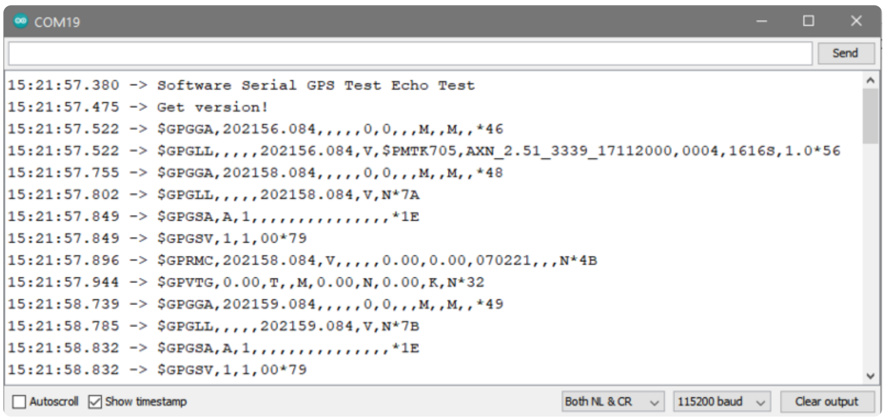
We assume you'll be using `Serial1` - check your board documentation to figure out which pins that is!



Upload and Check Output

Whichever version you decide to use, upload the sketch to the microcontroller. Then open up the serial monitor. This sketch simply reads data from the software serial or hardware serial port and outputs that to USB.

Open up the Arduino IDE Serial Console and make sure to set the Serial baud rate to 115200



```
15:21:57.380 -> Software Serial GPS Test Echo Test
15:21:57.475 -> Get version!
15:21:57.522 -> $GPGGA,202156.084,,,,,0,0,,,M,,M,,*46
15:21:57.522 -> $GPGLL,,,,,202156.084,V,$PMTK705,AXN_2.51_3339_17112000,0004,1616S,1.0*56
15:21:57.755 -> $GPGGA,202158.084,,,,,0,0,,,M,,M,,*48
15:21:57.802 -> $GPGLL,,,,,202158.084,V,N*7A
15:21:57.849 -> $GPGSA,A,1,,,,,,,,,,,,,,,*1E
15:21:57.849 -> $GPGSV,1,1,00*79
15:21:57.896 -> $GPRMC,202158.084,V,,,,,0.00,0.00,070221,,,N*4B
15:21:57.944 -> $GPVTG,0.00,T,,M,0.00,N,0.00,K,N*32
15:21:58.739 -> $GPGGA,202159.084,,,,,0,0,,,M,,M,,*49
15:21:58.785 -> $GPGLL,,,,,202159.084,V,N*7B
15:21:58.832 -> $GPGSA,A,1,,,,,,,,,,,,,,,*1E
15:21:58.832 -> $GPGSV,1,1,00*79
```

You can configure the GPS output you see by commenting/uncommenting lines in the setup() procedure. For example, we can ask the GPS to send different sentences, and change how often it sends data. 10 Hz (10 times a second) is the max speed, and is a lot of data. You may not be able to output "all data" at that speed because the 9600 baud rate is not fast enough.

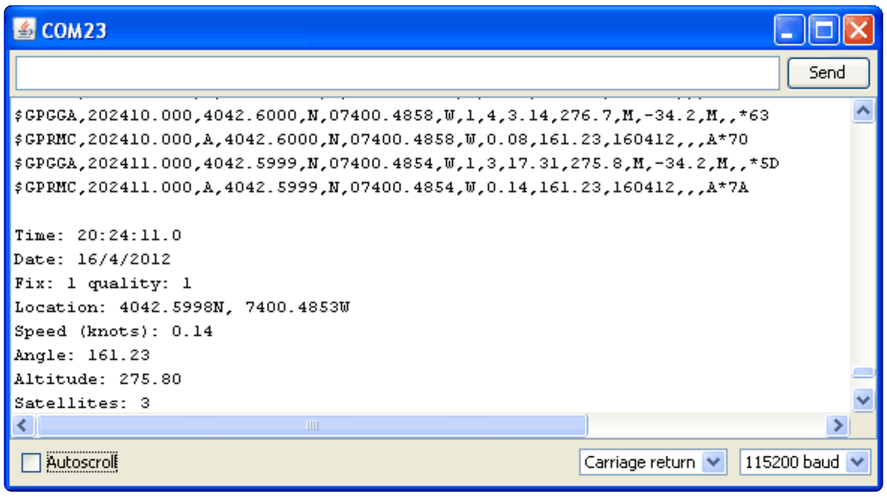
```
// You can adjust which sentences to have the module emit, below

  // uncomment this line to turn on RMC (recommended minimum) and GGA (fix data)
including altitude
  GPS.sendCommand(PMTK_SET_NMEA_OUTPUT_RMCGGA);
  // uncomment this line to turn on only the "minimum recommended" data for high
update rates!
  //GPS.sendCommand(PMTK_SET_NMEA_OUTPUT_RMCONLY);
  // uncomment this line to turn on all the available data - for 9600 baud you'll
want 1 Hz rate
  //GPS.sendCommand(PMTK_SET_NMEA_OUTPUT_ALLDATA);

  // Set the update rate
  // 1 Hz update rate
  //GPS.sendCommand(PMTK_SET_NMEA_UPDATE_1HZ);
  // 5 Hz update rate- for 9600 baud you'll have to set the output to RMC or RMCGGA
only (see above)
  GPS.sendCommand(PMTK_SET_NMEA_UPDATE_5HZ);
  // 10 Hz update rate - for 9600 baud you'll have to set the output to RMC only
(see above)
  //GPS.sendCommand(PMTK_SET_NMEA_UPDATE_10HZ);
```

In general, we find that most projects only need the RMC and GGA NMEA's so you don't need ALLDATA unless you have some need to know satellite locations.

Breakout Arduino Parsing

Since all GPS's output NMEA sentences and often for our projects we need to extract the actual data from them, we've simplified the task tremendously when using the Adafruit GPS library. By having the library read, store and parse the data in a background interrupt it becomes trivial to query the library and get the latest updated information without any icky parsing work.

Open up the File→Examples→Adafruit_GPS→GPS_HardwareSerial_Parsing or GPS_SoftwareSerial_Parsing sketch and upload it to the microcontroller. Use the same wiring as in the previous page. Then open up the serial monitor.



```
$GPGGA,202410.000,4042.6000,N,07400.4858,W,1,4,3.14,276.7,M,-34.2,M,,*63
$GPRMC,202410.000,A,4042.6000,N,07400.4858,W,0.08,161.23,160412,,,A*70
$GPGGA,202411.000,4042.5999,N,07400.4854,W,1,3,17.31,275.8,M,-34.2,M,,*5D
$GPRMC,202411.000,A,4042.5999,N,07400.4854,W,0.14,161.23,160412,,,A*7A

Time: 20:24:11.0
Date: 16/4/2012
Fix: 1 quality: 1
Location: 4042.5998N, 7400.4853W
Speed (knots): 0.14
Angle: 161.23
Altitude: 275.80
Satellites: 3
```

You'll only get correct time and location data if the GPS has a fix. Make sure it has an external antenna outside or is very close to a window and pointing up, to get GPS data! You'll usually get time information before location since you don't need as many satellites.

In this sketch, we call GPS.read() constantly in the main loop (if you can, get this to run once a millisecond in an interrupt). Then in the main loop we can ask if a new chunk of data has been received by calling GPS.newNMEAreceived(), if this returns true then we can ask the library to parse that data with GPS.parse(GPS.lastNMEA()).

We do have to keep querying and parsing in the main loop - its not possible to do this in an interrupt because then we'd be dropping GPS data by accident.

Once data is parsed, we can just ask for data from the library like GPS.day, GPS.mont h and GPS.year for the current date. GPS.fix will be 1 if there is a fix, 0 if there is none. If we have a fix then we can ask for GPS.latitude, GPS.longitude, GPS.speed (in knots, not mph or k/hr!), GPS.angle, GPS.altitude (in centimeters) and GPS.satellites (number of satellites)

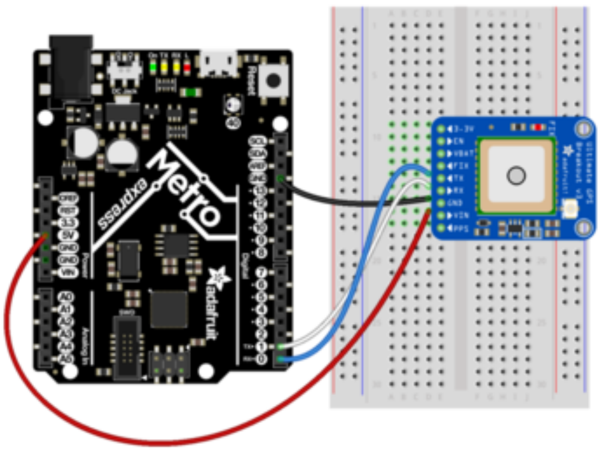
This should make it much easier to have location-based projects. We suggest keeping the update rate at 1Hz and request that the GPS only output RMC and GGA as the parser does not keep track of other data anyways.

CircuitPython & Python Setup

You can easily use a GPS module with Python or CircuitPython code in addition to Arduino. Python code is well suited for parsing and processing the text output from GPS modules, and this Adafruit CircuitPython GPS () module handles most of the work for you!

CircuitPython MicroController Wiring

First make sure to wire up the GPS module to your CircuitPython board so that the hardware UART pins are used. Here's an example with the Metro M0 Express:

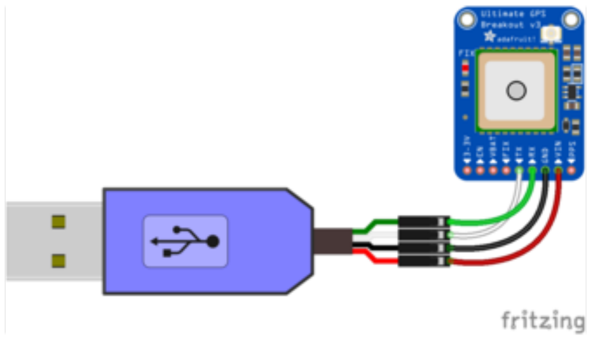


Board 5V or 3.3V to GPS module VIN.
Board GND to GPS module GND.
Board serial TX to GPS module RX.
Board serial RX to GPS module TX.

Python Computer Wiring

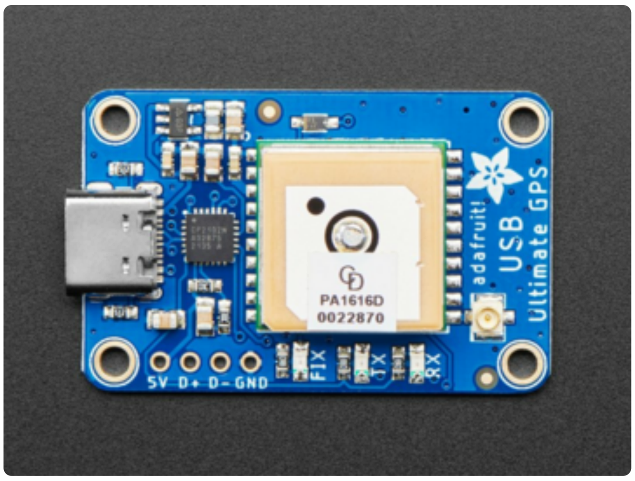
Since there's dozens of Linux computers/boards you can use we will show wiring for Raspberry Pi. For other platforms, please visit the guide for CircuitPython on Linux to see whether your platform is supported ().

Here you have two options: An external USB-to-serial converter, or the built-in UART on the Pi's TX/RX pins. Here's an example of wiring up the USB-to-serial converter ():



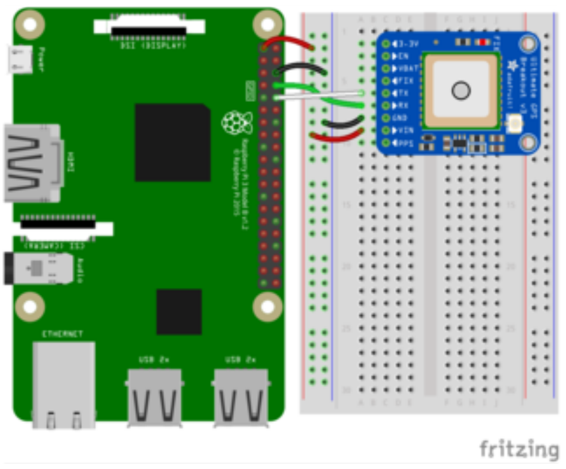
GPS Vin to USB 5V or 3V (red wire on USB console cable)
GPS Ground to USB Ground (black wire)
GPS RX to USB TX (green wire)
GPS TX to USB RX (white wire)

For single board computers other than the Raspberry Pi, the serial port may be tied to the console or not be available to the user. Please see the board documentation to see how the serial port may be used



You can also skip the USB console cable, and just plug a USB C or Micro B cable directly from your computer to the Ultimate GPS USB

Here's an example using the Pi's built-in UART:



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)

If you want to use the built-in UART, you'll need to disable the serial console and enable the serial port hardware in raspi-config. See the UART/Serial section of the CircuitPython on Raspberry Pi guide () for detailed instructions on how to do this.

CircuitPython Installation of GPS Library

Next you'll need to install the Adafruit CircuitPython GPS () library on your CircuitPython board. Remember this module is for Adafruit CircuitPython firmware and not MicroPython.org firmware!

First make sure you are running the latest version of Adafruit CircuitPython () for your board.

Next you'll need to install the necessary libraries to use the hardware. Carefully follow the steps to find and install this library () from Adafruit's CircuitPython library bundle ().

- adafruit_gps.mpy

You can also download the adafruit_gps.mpy file from the Adafruit CircuitPython GPS releases page ().

Before continuing make sure your board's lib folder has the adafruit_gps.mpy file copied over.

Python Installation of GPS Library

You'll need to install the Adafruit_Blinka library that provides the CircuitPython support in Python. This may require 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 ()!

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

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

CircuitPython & Python UART Usage

To demonstrate the usage of the GPS module in CircuitPython using UART, let's look at a complete program example, the gps_simpletest.py file from the module's examples.

CircuitPython Microcontroller

With a CircuitPython microcontroller, save this file as code.py on your board, then open the serial console to see its output.

Linux/Computer/Raspberry Pi with Python

If you're running gps_simpletest.py on the Raspberry Pi (or any computer), you'll have to make some changes.

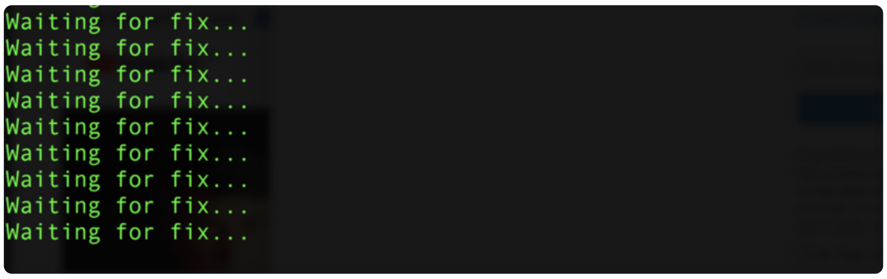
On the Raspberry Pi, comment out the `uart = busio(...)` line, and uncomment the `import serial` and `uart = serial.Serial(...)` lines, changing `/dev/ttyUSB0` to the appropriate serial port. Now you can run the program with the following command:

```
python3 gps_simpletest.py
```

Example Parsing Code

```
# 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.
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()  # uses board.SCL and board.SDA
# i2c = board.STEMMA_I2C()  # For using the built-in STEMMA QT connector on a
microcontroller

# 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"PMTK314,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0")
# Turn on just minimum info (RMC only, location):
# gps.send_command(b'PMTK314,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0')
# Turn off everything:
# gps.send_command(b'PMTK314,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0')
# Turn on everything (not all of it is parsed!)
# gps.send_command(b'PMTK314,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0')

# 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(
            "Precise Latitude: {:2.}{:2.4f} degrees".format(
                gps.latitude_degrees, gps.latitude_minutes
            )
        )
        print(
            "Precise Longitude: {:2.}{:2.4f} degrees".format(
                gps.longitude_degrees, gps.longitude_minutes
            )
        )
        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))
```

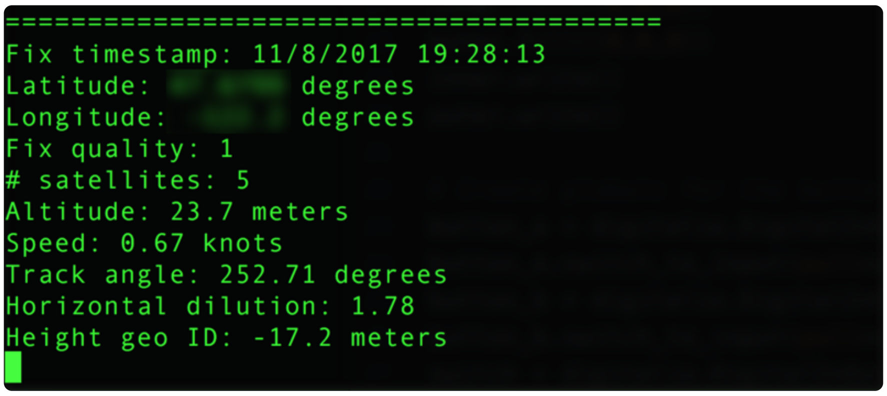
When the code runs it will print a message every second, either an update that it's still waiting for a GPS fix:



```
Waiting for fix...
Waiting for fix...
Waiting for fix...
Waiting for fix...
Waiting for fix...
Waiting for fix...
Waiting for fix...
Waiting for fix...
Waiting for fix...
```

Note: Due to the antenna being built in, the PA1010D Mini GPS module may need a more unobstructed view of the sky than other GPS modules with eternal antennae. With any GPS module, if you are having trouble getting a fix, try moving it to a more ideal location.

Once a fix has been established, it will print details about the current location and other GPS data:



```
========================================
Fix timestamp: 11/8/2017 19:28:13
Latitude: [redacted] degrees
Longitude: [redacted] degrees
Fix quality: 1
# satellites: 5
Altitude: 23.7 meters
Speed: 0.67 knots
Track angle: 252.71 degrees
Horizontal dilution: 1.78
Height geo ID: -17.2 meters
```

Let's look at the code in a bit more detail to understand how it works. First the example needs to import a few modules like the built-in `busio` and `board` modules that access serial ports and other hardware:

```
import board
import busio
import time
```

Next the GPS module is imported:

```
import adafruit_gps
```

Now a serial UART () is created and connected to the serial port pins the GPS module will use, this is the low level transport layer to communicate with the GPS module:

```
# 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=3000)

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

Once a `UART` object is available with a connected GPS module you can create an instance of the GPS parsing class. You need to pass this class the `UART` instance and it will internally read new data from the GPS module connected to it:

```
gps = adafruit_gps.GPS(uart)
```

GPS Example Code Explained

Before reading GPS data the example configures the module by sending some custom NMEA GPS commands () that adjust the amount and rate of data. Read the comments to see some options for adjust the rate and amount of data, but typically you want the defaults of core location info at a rate of once a second:

```
# 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'PMTK314,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0')
# Turn on just minimum info (RMC only, location):
#gps.send_command(b'PMTK314,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0')
# Turn off everything:
#gps.send_command(b'PMTK314,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0')
# Tuen on everything (not all of it is parsed!)
#gps.send_command(b'PMTK314,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0')

# 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')
```

If you want you can send other custom commands to the GPS module with the `send_command` function shown above. You don't need to worry about adding a NMEA checksum to your command either, the function will do this automatically (or not, set `add_checksum=False` as a parameter and it will skip the checksum addition).

Now we can jump into a main loop that continually updates data from the GPS module and prints out status. The most important part of this loop is calling the GPS update function:

```
    # 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()
```

Like the comments mention, you must call `update` every loop iteration and ideally multiple times a second. Each time you call `update`, it allows the GPS library code to read new data from the GPS module and update its state. Since the GPS module is always sending data you have to be careful to constantly read data or else you might start to lose data as buffers are filled.

You can check the `has_fix` property to see if the module has a GPS location fix, and if so there are a host of attributes to read like `latitude` and `longitude` (available in degrees):

```
        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: {} degrees'.format(gps.latitude))
        print('Longitude: {} 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.track_angle_deg 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:
```

Notice some of the attributes like `altitude_m` are checked to be `None` before reading. This is a smart check to put in your code, because those attributes are sometimes not sent by a GPS module. If an attribute isn't sent by the module it will be given a `None`/null value and attempting to print or read it in Python will fail. The core attributes of `latitude`, `longitude`, and `timestamp` are usually always available (if you're using the example as-is) but they might not be if you turn off those outputs with a custom NMEA command!

That's all there is to reading GPS location with CircuitPython code!

CircuitPython Datalogging

Datalogging Example

Another handy task with GPS is logging all the raw output of the GPS module to a file. This is useful if you're importing the GPS data into a tool like Google Earth which can process raw NMEA sentences. You can perform this datalogging very easily with CircuitPython.

To store data you'll need to choose one of two options:

- Wire up a SD card holder to your board's SPI bus, or use a board with SD card holder built-in like the Feather M0 Adalogger (). This is the recommended approach as it gives you a lot of space to store data and you can easily copy the data to your computer from the card.
- Store data in your board's internal filesystem. This requires a little more setup but allows you to save to a file on the internal filesystem of your CircuitPython board, right next to where code and other data files live. This is more limited because depending on your board you might only have a few kilobytes or megabytes of space available and GPS sentences will quickly add up (easily filling multiple megabytes within a few hours of logging).

Install SD Card Library

If you're storing data on a SD card you must ensure the SD card is wired to your board and you have installed the Adafruit SD card library. Luckily there's an entire guide to follow to learn about this process of connecting a SD card and installing the necessary library (). Be sure to carefully follow the guide so the card is connected, library installed, and you can confirm you're able to manually write data to the card from the Python prompt.

Enable Internal Filesystem Writes

If you're storing data on the internal filesystem you must carefully follow the steps in the CPU temperature logging guide to enable writing to internal storage (). If you're writing to a SD card skip these steps and move on to look at the datalogging code below. Edit the boot.py on your board (creating it if it doesn't exist) and add these lines:

```
import digitalio
import board
import storage

switch = digitalio.DigitalInOut(board.D5)
switch.direction = digitalio.Direction.INPUT
switch.pull = digitalio.Pull.UP

# If the D5 is connected to ground with a wire
# you can edit files over the USB drive again.
storage.remount("/", not switch.value)
```

Remember once you remount("/") you cannot edit code over the USB drive anymore! That means you can't edit boot.py which is a bit of a conundrum. So we configure the boot.py to selectively mount the internal filesystem as writable based on a switch or even just alligator clip connected to ground. Like the CPU temperature guide shows () . In this example we're using D5 but select any available pin.

This code will look at the D5 digital input when the board starts up and if it's connected to ground (use an alligator clip or wire, for example, to connect from D5 to board ground) it will disable internal filesystem writes and allow you to edit code over the USB drive as normal. Remove the alligator clip, reset the board, and the boot.py will switch to mounting the internal filesystem as writable so you can log images to it again (but not write any code!).

Remember when you enable USB drive writes (by connecting D5 to ground at startup) you cannot write files to the internal filesystem and any code in your main.py that attempts to do so (like the example below) will fail. Keep this in mind as you edit code--once you modify code you need to remove the alligator clip, reset the board to re-enable internal filesystem writes, and then watch the output of your program.

If you ever get stuck, you can follow the steps mentioned in https://learn.adafruit.com/cpu-temperature-logging-with-circuit-python/writing-to-the-filesystem to remove boot.py from the REPL if you need to go back and edit code!

Datalogging Example Code

The GPS library examples have a datalogging.py file you can edit and save as a code.py on your board:

```
# 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 sys

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"

# sdcardio and adafruit_sdcard are NOT supported on blinka. If you are using a
# Raspberry Pi or other single-board linux computer, the code will save the
# output to the path defined in LOG_FILE above.
if sys.platform != "linux":
    import storage

    SD_CS_PIN = board.D10  # CS for SD card using Adalogger Featherwing
    try:
        import sdcardio

        sdcard = sdcardio.SDCard(board.SPI, SD_CS_PIN)
    except ImportError:
        import adafruit_sdcard
        import digitalio

        sdcard = adafruit_sdcard.SDCard(
            board.SPI(),
            digitalio.DigitalInOut(SD_CS_PIN),
        )

    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()  # uses board.SCL and board.SDA
# i2c = board.STEMMA_I2C()  # For using the built-in STEMMA QT connector on a
microcontroller

# 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()
```

By default this example expects to log GPS NMEA sentences to a file on the internal storage system at /gps.txt. New sentences will be appended to the end of the file every time the example starts running.

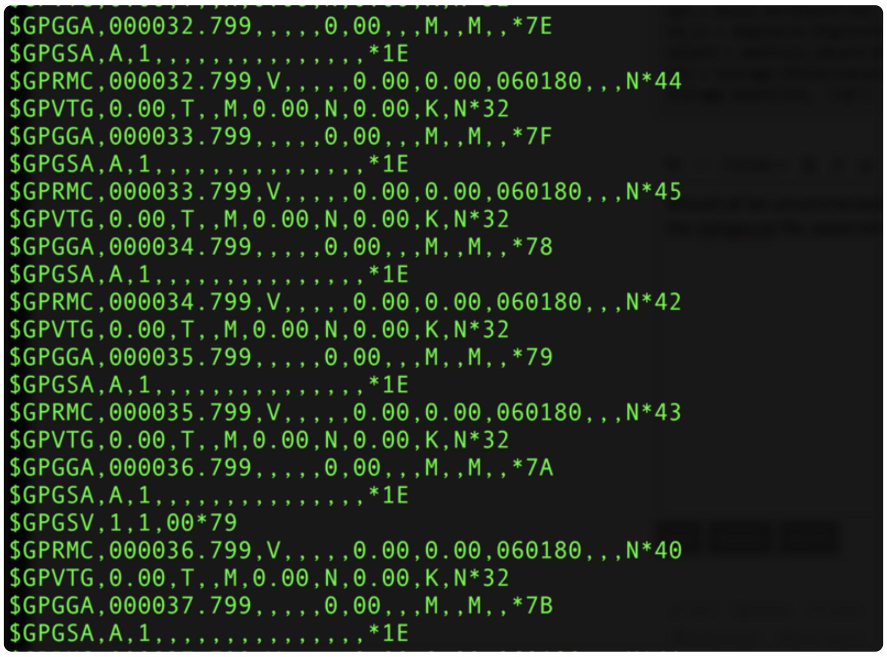
If you'd like to instead write to the SD card take note to uncomment the appropriate lines mentioned in the comments:

```
# 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
LOG_FILE = '/sd/gps.txt'     # Example for writing to SD card path /sd/gps.txt
```

And further below:

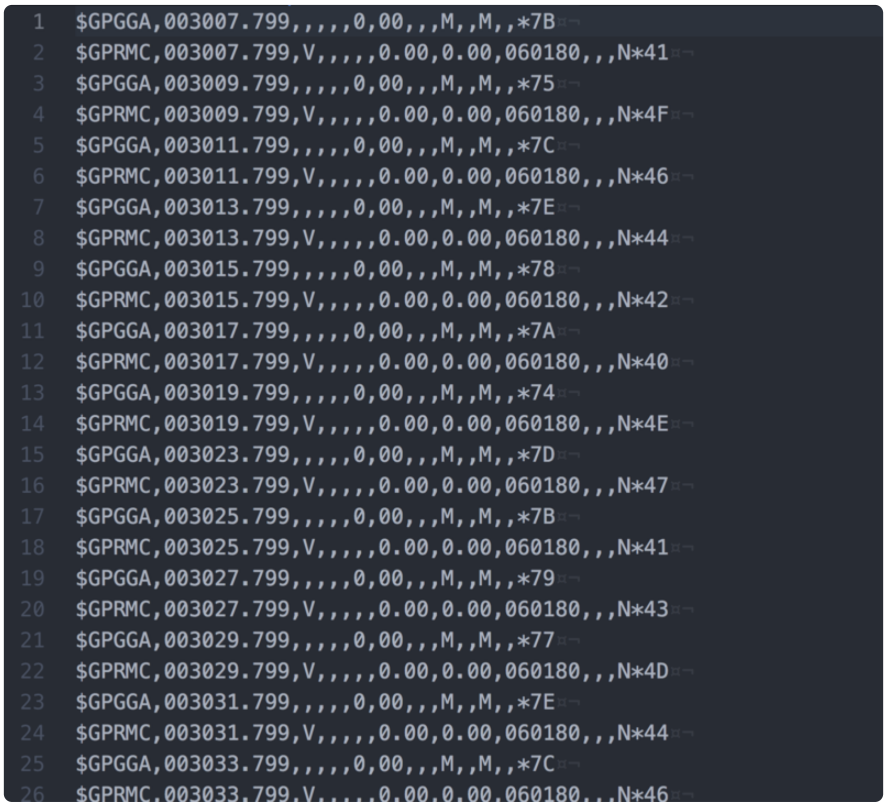
Should all be uncommented and look as above. This will configure the code to write GPS NMEA data to the /sd/gps.txt file, appending new data to the end of the file.

Once the example is running as a main.py on your board open the serial REPL and you should see the raw NMEA sentences printed out:



```
$GPGGA,000032.799,,,,,0,00,,,M,,M,,*7E
$GPGSA,A,1,,,,,,,,,,,,,,,*1E
$GPRMC,000032.799,V,,,,,0.00,0.00,060180,,,N*44
$GPVTG,0.00,T,,M,0.00,N,0.00,K,N*32
$GPGGA,000033.799,,,,,0,00,,,M,,M,,*7F
$GPGSA,A,1,,,,,,,,,,,,,,,*1E
$GPRMC,000033.799,V,,,,,0.00,0.00,060180,,,N*45
$GPVTG,0.00,T,,M,0.00,N,0.00,K,N*32
$GPGGA,000034.799,,,,,0,00,,,M,,M,,*78
$GPGSA,A,1,,,,,,,,,,,,,,,*1E
$GPRMC,000034.799,V,,,,,0.00,0.00,060180,,,N*42
$GPVTG,0.00,T,,M,0.00,N,0.00,K,N*32
$GPGGA,000035.799,,,,,0,00,,,M,,M,,*79
$GPGSA,A,1,,,,,,,,,,,,,,,*1E
$GPRMC,000035.799,V,,,,,0.00,0.00,060180,,,N*43
$GPVTG,0.00,T,,M,0.00,N,0.00,K,N*32
$GPGGA,000036.799,,,,,0,00,,,M,,M,,*7A
$GPGSA,A,1,,,,,,,,,,,,,,,*1E
$GPGSV,1,1,00*79
$GPRMC,000036.799,V,,,,,0.00,0.00,060180,,,N*40
$GPVTG,0.00,T,,M,0.00,N,0.00,K,N*32
$GPGGA,000037.799,,,,,0,00,,,M,,M,,*7B
$GPGSA,A,1,,,,,,,,,,,,,,,*1E
```

Check the gps.txt file (either under the root or /sd path depending on how you setup the example) in a text editor and you'll see the same raw NMEA sentences:



```
$GPGGA,003007.799,,,,,0,00,,,M,,M,,*7B
$GPRMC,003007.799,V,,,,,0.00,0.00,060180,,,N*41
$GPGGA,003009.799,,,,,0,00,,,M,,M,,*75
$GPRMC,003009.799,V,,,,,0.00,0.00,060180,,,N*4F
$GPGGA,003011.799,,,,,0,00,,,M,,M,,*7C
$GPRMC,003011.799,V,,,,,0.00,0.00,060180,,,N*46
$GPGGA,003013.799,,,,,0,00,,,M,,M,,*7E
$GPRMC,003013.799,V,,,,,0.00,0.00,060180,,,N*44
$GPGGA,003015.799,,,,,0,00,,,M,,M,,*78
$GPRMC,003015.799,V,,,,,0.00,0.00,060180,,,N*42
$GPGGA,003017.799,,,,,0,00,,,M,,M,,*7A
$GPRMC,003017.799,V,,,,,0.00,0.00,060180,,,N*40
$GPGGA,003019.799,,,,,0,00,,,M,,M,,*74
$GPRMC,003019.799,V,,,,,0.00,0.00,060180,,,N*4E
$GPGGA,003023.799,,,,,0,00,,,M,,M,,*7D
$GPRMC,003023.799,V,,,,,0.00,0.00,060180,,,N*47
$GPGGA,003025.799,,,,,0,00,,,M,,M,,*7B
$GPRMC,003025.799,V,,,,,0.00,0.00,060180,,,N*41
$GPGGA,003027.799,,,,,0,00,,,M,,M,,*79
$GPRMC,003027.799,V,,,,,0.00,0.00,060180,,,N*43
$GPGGA,003029.799,,,,,0,00,,,M,,M,,*77
$GPRMC,003029.799,V,,,,,0.00,0.00,060180,,,N*4D
$GPGGA,003031.799,,,,,0,00,,,M,,M,,*7E
$GPRMC,003031.799,V,,,,,0.00,0.00,060180,,,N*44
$GPGGA,003033.799,,,,,0,00,,,M,,M,,*7C
$GPRMC,003033.799,V,,,,,0.00,0.00,060180,,,N*46
```

Awesome! That's all there is to basic datalogging of NMEA sentences with a GPS module and CircuitPython!

Python Docs

Python Docs ()

Built In Logging

One of the nice things about the MTK3339 is the built in data-logger. This basic data-logging capability can store date, time, latitude, longitude and altitude data into a 64K flash chip inside. Its not a high resolution logger - it only logs once every 15 seconds when there is a fix - but for 99% of projects that want to track location, this can be a great low power way to log data - no SD card or other EEPROM required! It can store up to 16 hours of data.

The GPS module does require a microcontroller to 'kick start' the logger by requesting it to start. If power is lost it will require another 'kick' to start. If you already have some data in the FLASH, a new trace will be created (so you wont lose old data) and if you run out of space it will simply halt and not overwrite old data. Despite this annoyance, its still a very nice extra and we have some library support to help you use it

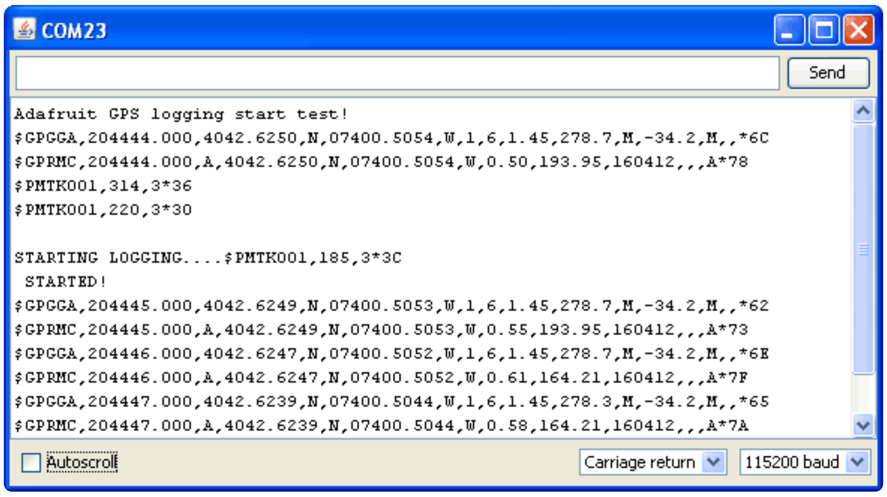
For more details check out the LOCUS (built-in-datalogging system) user guide ()

First, we should try getting the logger to run.Open up the File→Examples→Adafruit_GPS→locus_start sketch. This will demonstrate how to start the logger (called LOCUS)

The key part is here:

```
Serial.print("STARTING LOGGING....");
  if (GPS.LOCUS_StartLogger())
    Serial.println(" STARTED!");
  else
    Serial.println(" no response :(");
  delay(1000);
```

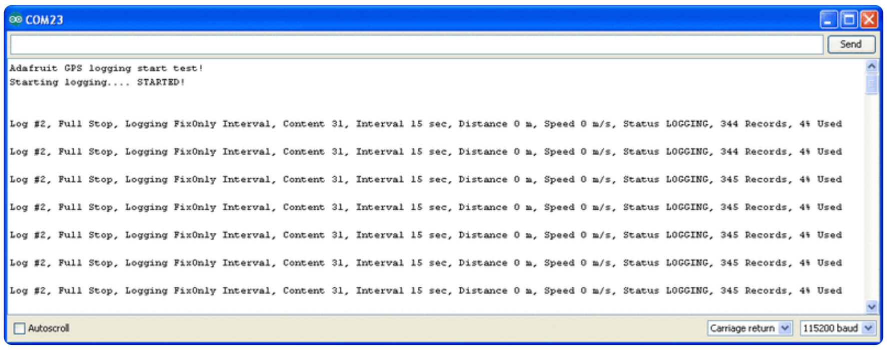
You should start the logger and then check the response:



```
Adafruit GPS logging start test!
$GPGGA,204444.000,4042.6250,N,07400.5054,W,1,6,1.45,278.7,M,-34.2,M,,*6C
$GPRMC,204444.000,A,4042.6250,N,07400.5054,W,0.50,193.95,160412,,,A*78
$PMTK001,314,3*36
$PMTK001,220,3*30

STARTING LOGGING....$PMTK001,185,3*3C
 STARTED!
$GPGGA,204445.000,4042.6249,N,07400.5053,W,1,6,1.45,278.7,M,-34.2,M,,*62
$GPRMC,204445.000,A,4042.6249,N,07400.5053,W,0.55,193.95,160412,,,A*73
$GPGGA,204446.000,4042.6247,N,07400.5052,W,1,6,1.45,278.7,M,-34.2,M,,*6E
$GPRMC,204446.000,A,4042.6247,N,07400.5052,W,0.61,164.21,160412,,,A*7F
$GPGGA,204447.000,4042.6239,N,07400.5044,W,1,6,1.45,278.3,M,-34.2,M,,*65
$GPRMC,204447.000,A,4042.6239,N,07400.5044,W,0.58,164.21,160412,,,A*7A
```

Logging Status

Once you've seen that the GPS is OK with logging, you can load up the status sketch which will also give you more data. Upload File→Examples→Adafruit_GPS→locus_stat us



```
Adafruit GPS logging start test!
Starting logging.... STARTED!


Log #2, Full Stop, Logging FixOnly Interval, Content 31, Interval 15 sec, Distance 0 m, Speed 0 m/s, Status LOGGING, 344 Records, 4% Used

Log #2, Full Stop, Logging FixOnly Interval, Content 31, Interval 15 sec, Distance 0 m, Speed 0 m/s, Status LOGGING, 344 Records, 4% Used

Log #2, Full Stop, Logging FixOnly Interval, Content 31, Interval 15 sec, Distance 0 m, Speed 0 m/s, Status LOGGING, 345 Records, 4% Used

Log #2, Full Stop, Logging FixOnly Interval, Content 31, Interval 15 sec, Distance 0 m, Speed 0 m/s, Status LOGGING, 345 Records, 4% Used

Log #2, Full Stop, Logging FixOnly Interval, Content 31, Interval 15 sec, Distance 0 m, Speed 0 m/s, Status LOGGING, 345 Records, 4% Used

Log #2, Full Stop, Logging FixOnly Interval, Content 31, Interval 15 sec, Distance 0 m, Speed 0 m/s, Status LOGGING, 345 Records, 4% Used

Log #2, Full Stop, Logging FixOnly Interval, Content 31, Interval 15 sec, Distance 0 m, Speed 0 m/s, Status LOGGING, 345 Records, 4% Used
```

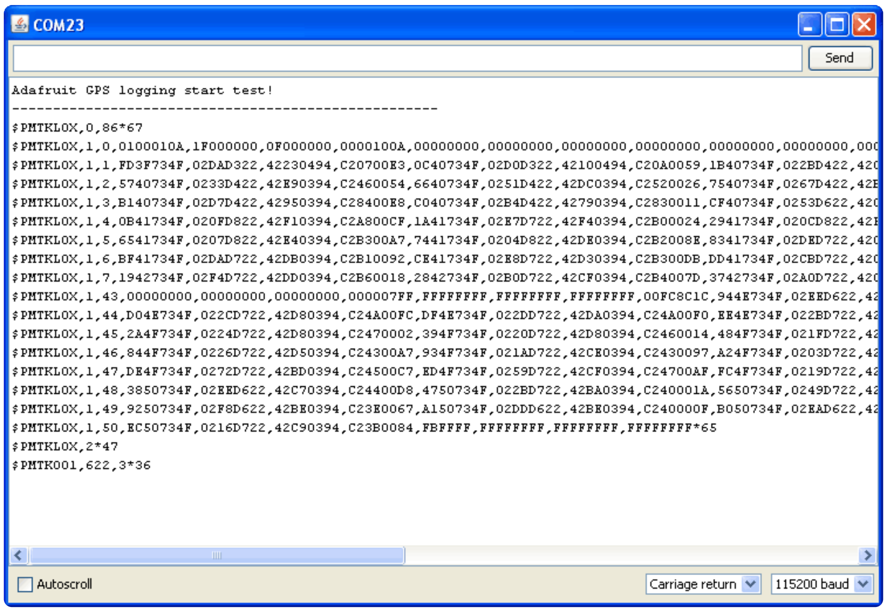
This output gives you some more information. the first entry is the Log #. This is how many log traces are in the memory. Every time you start and save data, a new log is made. Full Stop means that once the logger has run out of memory it will stop. Next the output indicates that we are logging only during fix data and at set intervals, with an interval delay of 15 seconds. We are not logging based on distance or speed. The current status is LOGGING (active), there's also the number of records we've stored. Each record is a timestamped location. We log once every 15 seconds, you can see the records increment from 344 to 345 here. Lastly, we can see how much of the internal flash storage is used, only 4% at this point

In real use, you'll probably want to start the loggging and then have your microcontroller go to sleep to reserve power, waking up once in a while to check up on the logging status.

Downloading Data

Finally, once we're done logging we need to extract the data. To do this we need to first get the raw data out of the FLASH and then decode the sentences. UploadFile→Examples→Adafruit_GPS→locus_dump to the Arduino and open up the serial monitor

PLEASE NOTE: Asking the Arduino, with 2K RAM buffer to handle 64KB of FLASH data and spit it out from the GPS can sometimes over-tax the processor. If you are having hiccups, check the GPS tool instructions below



```
Adafruit GPS logging start test!
----------------------------------------------------
$PMTKLOX,0,86*67
$PMTKLOX,1,0,0100010A,1F000000,0F000000,0000100A,00000000,00000000,00000000,00000000,00000000,00000000,000
$PMTKLOX,1,1,FD3F734F,02DAD322,42230494,C20700E3,0C40734F,02D0D322,42100494,C20A0059,1B40734F,022BD422,420
$PMTKLOX,1,2,5740734F,0233D422,42E90394,C2460054,6640734F,0251D422,42DC0394,C2520026,7540734F,0267D422,42E
$PMTKLOX,1,3,B140734F,02D7D422,42950394,C28400E8,C040734F,02B4D422,42790394,C2830011,CF40734F,0253D622,420
$PMTKLOX,1,4,0B41734F,020FD822,42F10394,C2A800CF,1A41734F,02E7D722,42F40394,C2B00024,2941734F,020CD822,42E
$PMTKLOX,1,5,6541734F,0207D822,42E40394,C2B300A7,7441734F,0204D822,42DE0394,C2B2008E,8341734F,02DED722,420
$PMTKLOX,1,6,BF41734F,02DAD722,42DB0394,C2B10092,CE41734F,02E8D722,42D30394,C2B300DB,DD41734F,02CBD722,420
$PMTKLOX,1,7,1942734F,02F4D722,42DD0394,C2B60018,2842734F,02B0D722,42CF0394,C2B4007D,3742734F,02A0D722,420
$PMTKLOX,1,43,00000000,00000000,00000000,000007FF,FFFFFFFF,FFFFFFFF,FFFFFFFF,00FC8C1C,944E734F,02EED622,42
$PMTKLOX,1,44,D04E734F,022CD722,42D80394,C24A00FC,DF4E734F,022DD722,42DA0394,C24A00F0,EE4E734F,022BD722,42
$PMTKLOX,1,45,2A4F734F,0224D722,42D80394,C2470002,394F734F,0220D722,42D80394,C2460014,484F734F,021FD722,42
$PMTKLOX,1,46,844F734F,0226D722,42D50394,C24300A7,934F734F,021AD722,42CE0394,C2430097,A24F734F,0203D722,42
$PMTKLOX,1,47,DE4F734F,0272D722,42BD0394,C24500C7,ED4F734F,0259D722,42CF0394,C24700AF,FC4F734F,0219D722,42
$PMTKLOX,1,48,3850734F,02EED622,42C70394,C24400D8,4750734F,022BD722,42BA0394,C240001A,5650734F,0249D722,42
$PMTKLOX,1,49,9250734F,02F8D622,42BE0394,C23E0067,A150734F,02DDD622,42BE0394,C240000F,B050734F,02EAD622,42
$PMTKLOX,1,50,EC50734F,0216D722,42C90394,C23B0084,FBFFFF,FFFFFFFF,FFFFFFFF,FFFFFFFF*65
$PMTKLOX,2*47
$PMTK001,622,3*36
```

Copy and paste all the text after the —-'s (starting with $PMTKLOX,0,86*67 and ending with $PMTK001,622,3*36) then paste it into the box located on this page ()

OR

you can try this python tool that don has kindly donated () to the community!

Using the GPS Tool

If you are having difficulty with the Arduino/javascript tool, you can also try using the GPS tool. The tool runs only under Windows but it is very powerful.

Connect the GPS module to an Arduino (connected with the Direct Wiring example) () , FTDI adapter or other TTL converter and download the GPS Tool () - connect to the

GPS via the COM port of the Arduino/FTDI/TTL cable. You can then query, dump and delete the log memory

LOCUS Parser

LOCUS Parser ()

Resources

Datasheets

- MTK3329/MTK3339 command set sheet () for changing the fix data rate, baud rate, sentence outputs, etc!
- PMTK 'complete' data ()sheet (like the above but with even more commands)
- Datasheet for the PA6B (MTK3329) GPS module itself ()
- Datasheet for the PA6C (MTK3339) GPS module itself ()
- Datasheet for the PA6H (MTK3339) GPS module itself ()
- MT3339 GPS PC Tool (windows only) () and the PC Tool manual ()
- Sample code and spec sheet for the LOCUS built-in logger ()
- LOCUS (built-in-datalogging system) user guide ()
- Mini GPS tool (windows only) ()

More reading:

- Trimble's GPS tutorial ()
- Garmin's GPS tutorial ()

Adafruit GPS Library for Arduino

https://github.com/adafruit/Adafruit-GPS-Library/ ()

EPO files for AGPS use

Data format for EPO files ()

MTK_EPO_Nov_12_2014.zip

F.A.Q.

Can the Ultimate GPS be used for High Altitude? How can I know?

Modules shipped in 2013+ (and many in the later half of 2012) have firmware that has been tested by simulation at the GPS factory at 40km.

You can tell what firmware you have by sending the firmware query command $PMTK605*31 (you can use the echo demo to send custom sentences to your GPS)

If your module replies with AXN_2.10_3339_2012072601 5223 that means you have version #5223 which is the 40Km-supported firmware version. If the number is higher then 5223 then its even more recent, and should include the 40Km support as well

HOWEVER these modules are not specifically designed for high-altitude balloon use. People have used them successfully but since we (at Adafruit) have not personally tested them for hi-alt use, we do not in any way guarantee they are suitable for high altitude use.

Please do not ask us to 'prove' that they are good for high altitude use, we do not have any way to do so

If you want to measure high altitude with a GPS, please find a module that can promise/guarantee high altitude functionality

Is the Ultimate GPS affected by the 2019 Week Rollover issue?

The ultimate GPS (all firmware versions from 20110922_GTOP_EVK01_A2.10 and higher - any sold after 2011) have been tested to work fine through 2019.

They do not pass the 2038 rollover test, so you may need to update the firmware between now and 2038. This does not affect the 2019 rollover (there's one every ~20 years)

OK I want the latest firmware!

Here is the binary of the 5632 firmware (), you can use this tool to upload it using an FTDI or USB-TTL cable (or direct wiring with FTDI) (). We do not have a tutorial for updating the firmware, if you update the firmware and somehow brick the GPS, we do not offer replacements! Keep this in mind before performing the update process!

I've adapted the example code and my GPS NMEA sentences are all garbled and incomplete!

We use SoftwareSerial to read from the GPS, which is 'bitbang' UART support. It isn't super great on the Arduino and does work but adding too many delay()s and not calling the GPS data parser enough will cause it to choke on data.

If you are using Leonardo (or Micro/Flora/ATmega32u4) or Mega, consider using a HardwareSerial port instead of SoftwareSerial!

My GPS is giving me data but the location is wrong!

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.)

How come I can't get the GPS to output at 10Hz?

The default baud rate to the GPS is 9600 - this can only do RMC messages at 10Hz. If you want more data output, you can increase the GPS baud rate (to 57600 for example) or go with something like 2 or 5Hz. There is a trade off with more data you want the GPS to output, the GPS baud rate, Arduino buffer/processing capability and update rate!

Experimentation may be necessary to get the optimal results. We suggest RMC only for 10Hz since we've tested it.

How come I can't set the RTC with the Adafruit RTC library?

The real time clock in the GPS is NOT 'writable' or accessable otherwise from the Arduino. Its in the GPS only! Once the battery is installed, and the GPS gets its first data reception from satellites it will set the internal RTC. Then as long as the battery is installed, you can read the time from the GPS as normal. Even without a proper "gps fix" the time will be correct.

The timezone cannot be changed, so you'll have to calculate local time based on

UTC!

Do all GPS modules emit PPS pulses at the same time?

Under ideal conditions, GPS modules emit a PPS signal within 10ns of the beginning of each GPS second. That's only a best-case value though.

In practice, each GPS module's sync to the GPS clock system depends on the quality of the fix, how long the GPS module has had a fix, and the group of satellites the module uses for its fix. We've observed offsets of about 300ns between modules that have just acquired a fix, improving to less than 100ns after the modules have had a good fix (Signal-to-Noise ratio higher than 20 for the satellites the modules use for their fix) for ten minutes.

When two GPS modules used the same group of satellites for their fix, there was less than 30ns of offset between PPS pulses as soon as the modules acquired a fix.

Why am I not seeing anything on the PPS pin?

The PPS pin only starts outputting after a 3D fix. In our testing, it truly wants a 3D fix, not just a 2D fix. Therefore, the PPS output may not happen even though the FIX LED and pin are indicating a fix.

You can check the current mode via the $GPGSA sentence. The second value must be a 3, as shown below:

```
$GPGSA,A,3,19,28,14,18,27,22,31,39,,,,,1.7,1.0,1.3*35
```

If the value is a 1 (Fix not available) or a 2 (2D), then the PPS pin may not output.

How can I read the PPS signal on the Ultimate GPS USB?

The PPS line is tied to the serial port RI (Ring Indicator) pin. You can read this with your serial-port interface code. for example here's a version in Python using pyserial:

```
import serial

ser = serial.Serial('/dev/ttyS34')  # open serial port
print(ser.name)           # check which port was really used
```

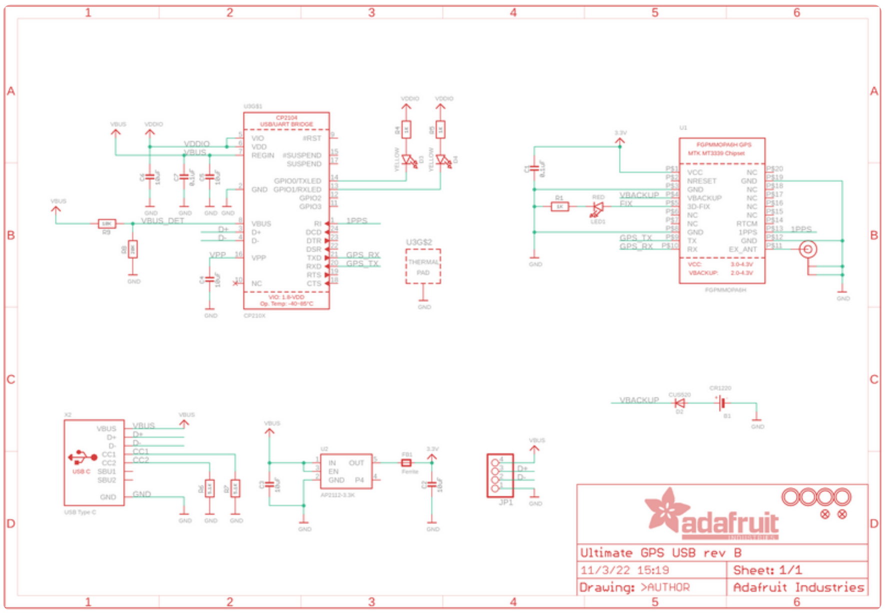
```
last_ri = ser.ri
while True:
    if ser.ri != last_ri:
        last_ri = ser.ri
        if last_ri:
            print("\n---------- Pulse high ----------")
        else:
            print("\n---------- Pulse low ----------")
    if ser.in_waiting:
        print(ser.read(ser.in_waiting).decode('utf-8'), end="")
```

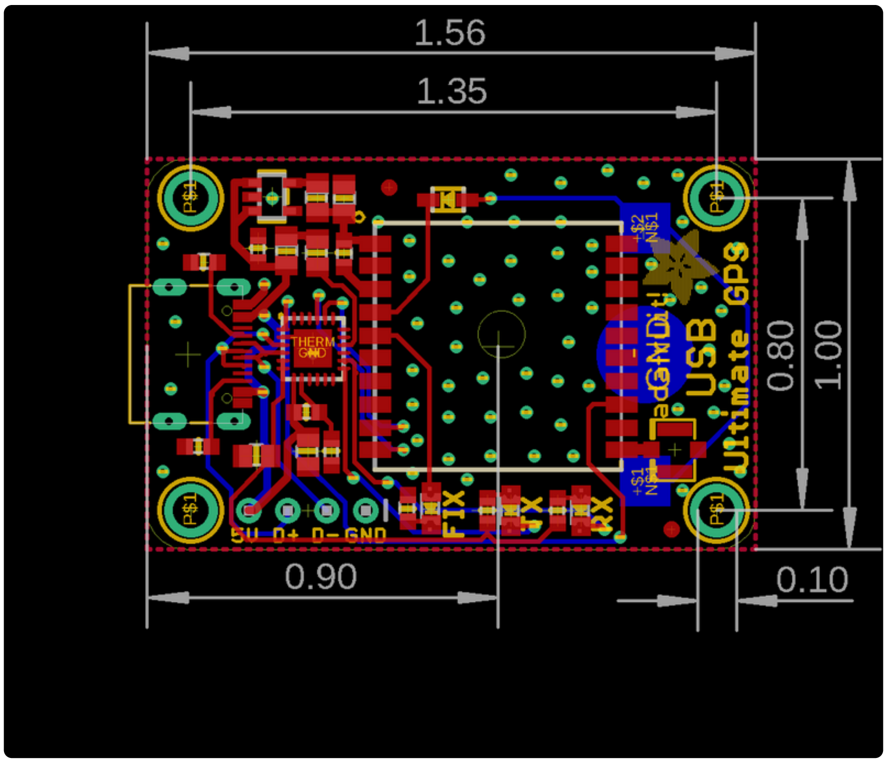
Downloads

Files

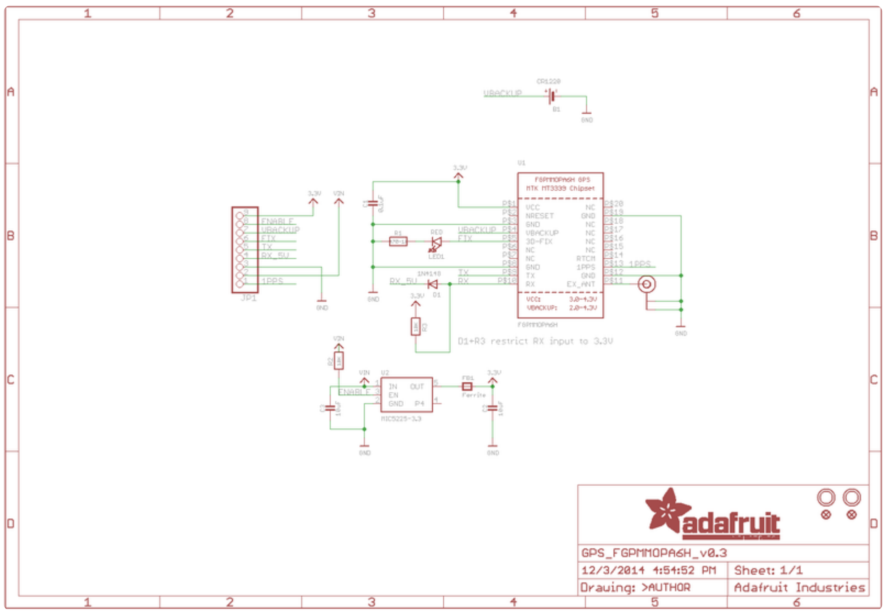
- MTK3329/MTK3339 command set sheet () for changing the fix data rate, baud rate, sentence outputs, etc!
- LOCUS (built-in-datalogging system) user guide ()
- Datasheet for the PA6B (MTK3329) GPS module itself - used in version 1 of this module ()
- Datasheet for the PA6C (MTK3339) GPS module itself - used in version 2 of this module ()
- Datasheet for the PA6H (MTK3339) GPS module itself - used in version 3 of this module ()
- Datasheet for the PA1616S (MTK3339) GPS module itself - used in version 3.1 of this module ()
- MT3339 GPS PC Tool (windows only) () and the PC Tool manual ()
- Mini GPS tool (windows only) ()
- EagleCAD PCB files on GitHub ()
- Fritzing object in the Adafruit Fritzing Library ()

Ultimate GPS USB-C Schematic and Fab Print



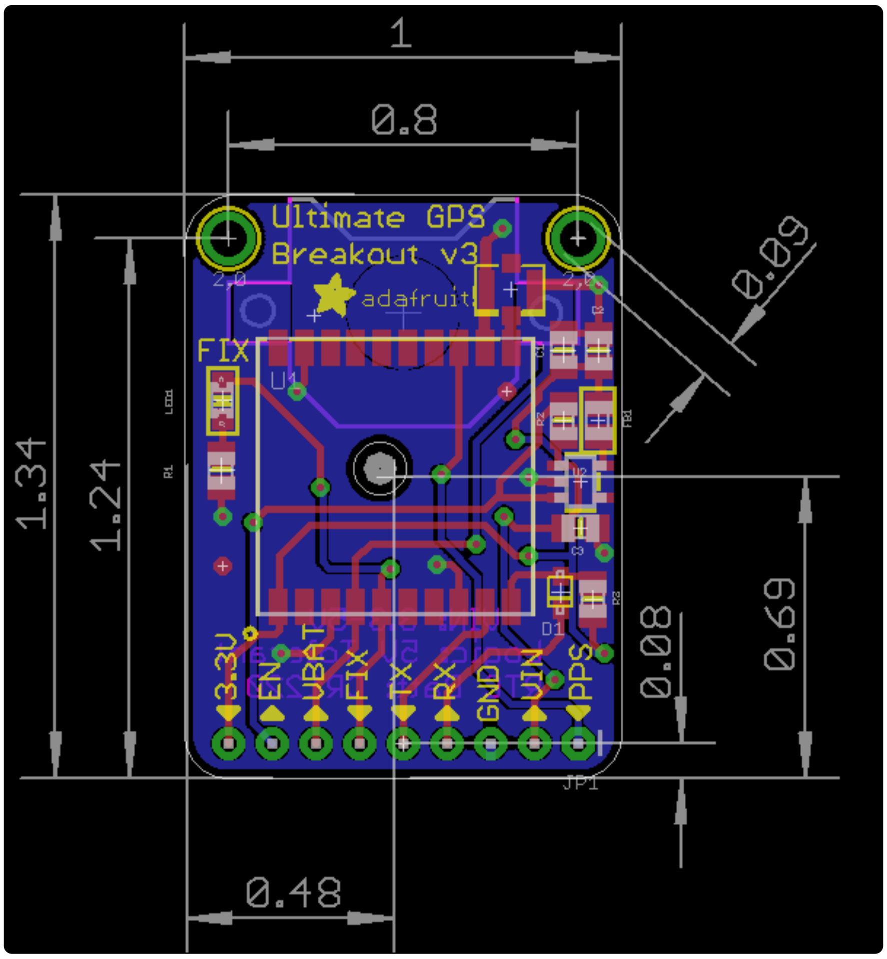


Ultimate GPS v3 Schematic

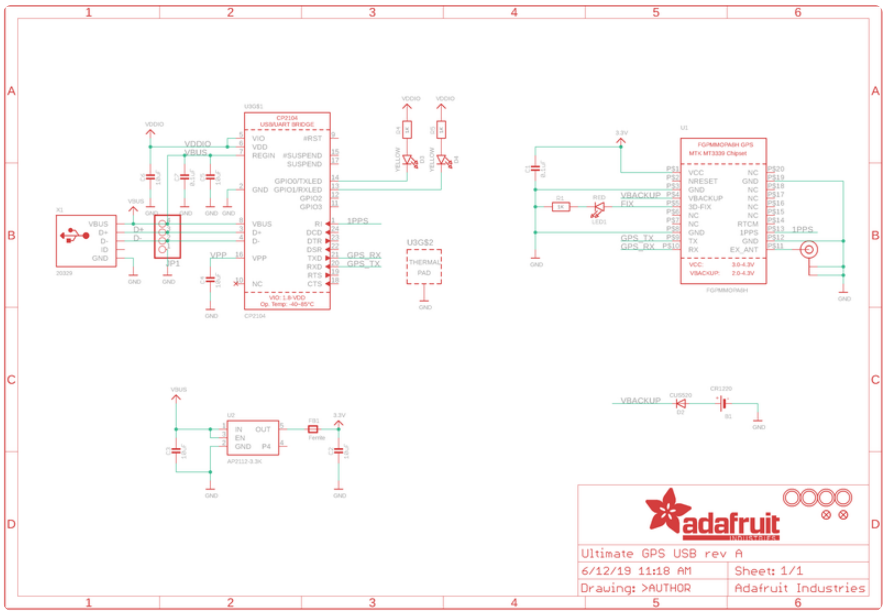


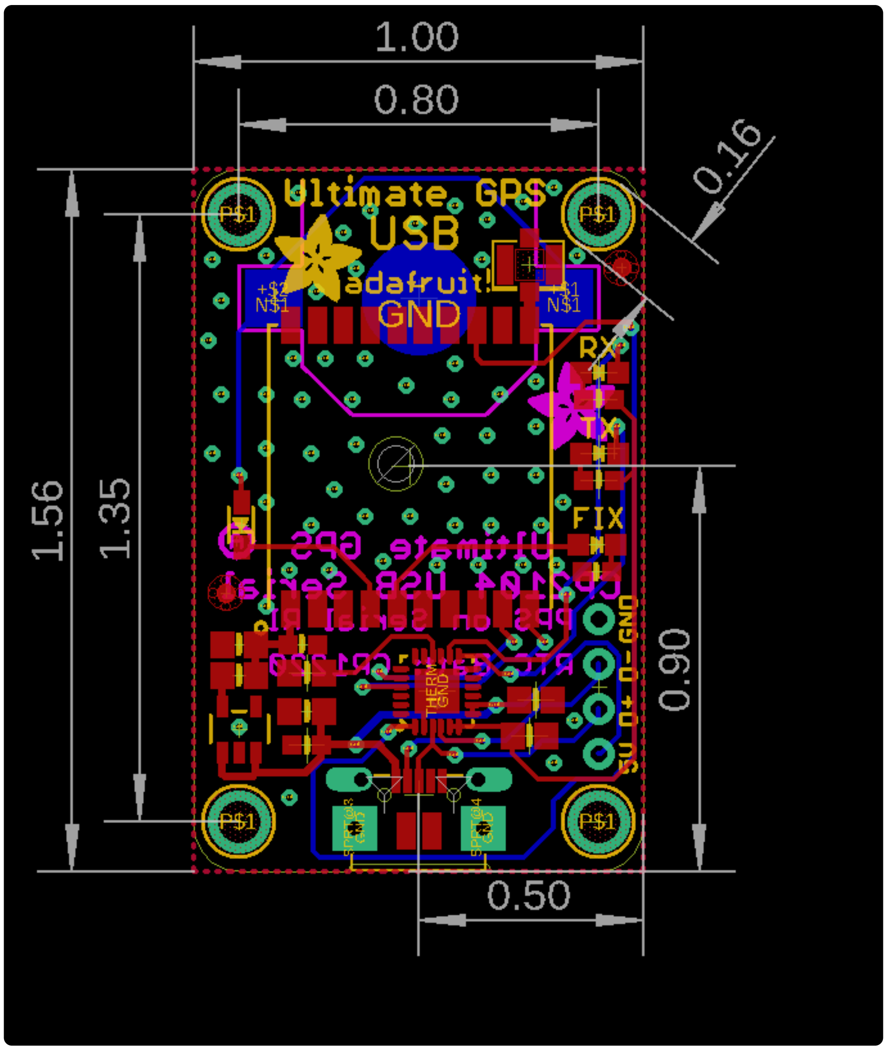
Ultimate GPS Fabrication Print

Dimensions in Inches



Ultimate GPS USB Micro B Version Schematic And Fab Print




1.00
0.80
0.16
1.56
1.35
0.90
0.50
Ultimate GPS
USB
adafruit
GND
RX
TX
FIX
5V D+ D- GND
P$1