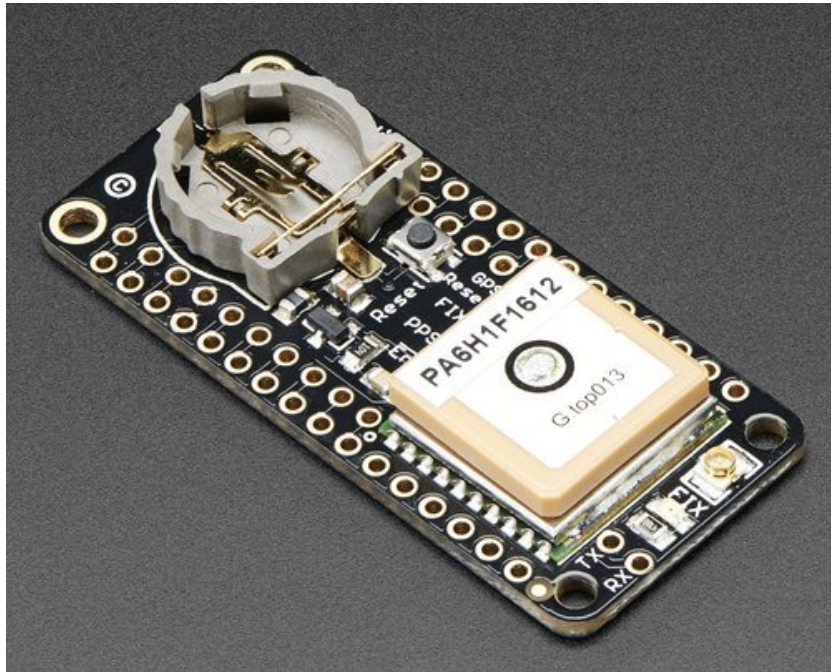


□

Adafruit Ultimate GPS featherwing

Created by lady ada

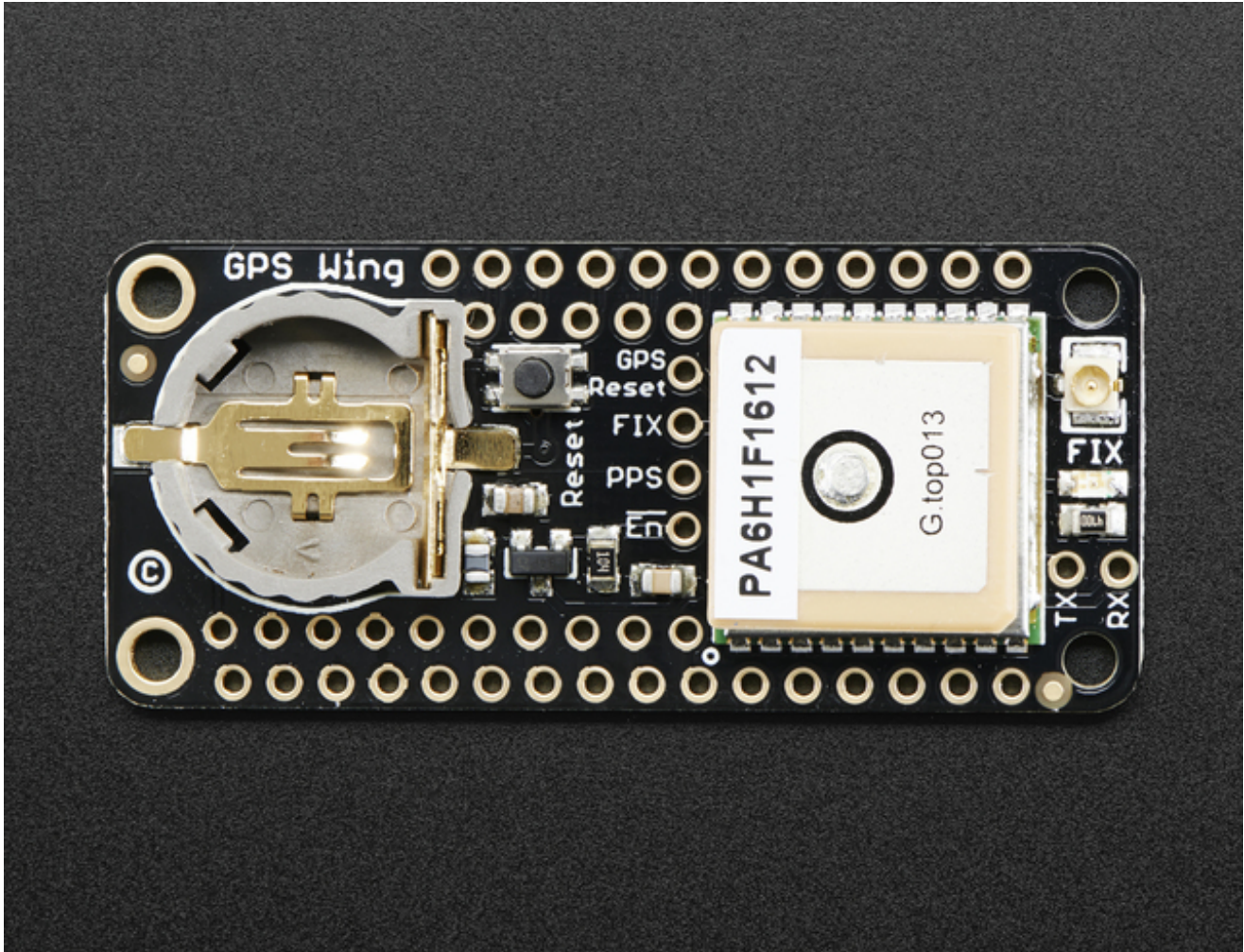


Last updated on 2017-03-21 08:20:13 PM UTC

Guide Contents

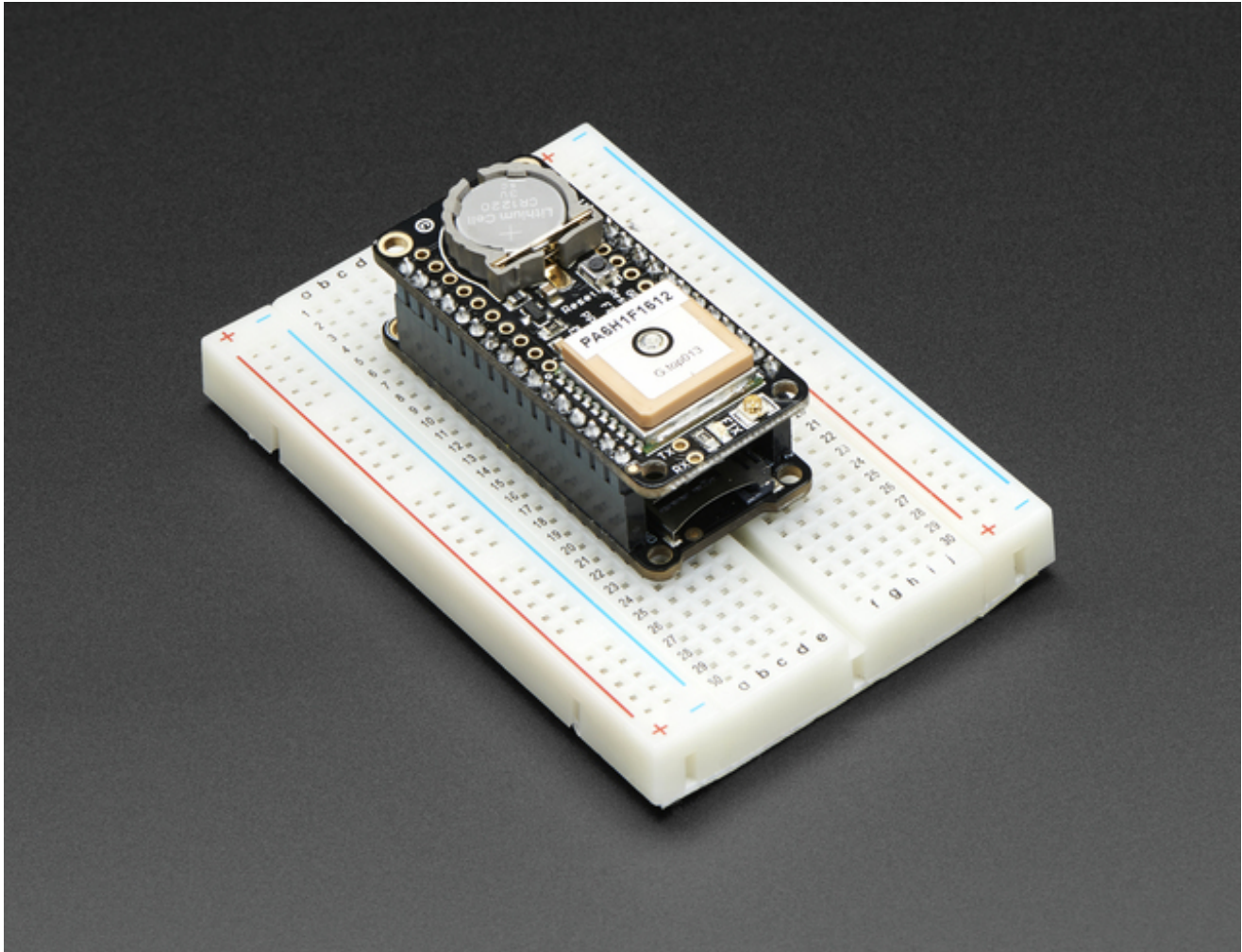
Guide Contents	2
Overview	3
Pinouts	8
Power Pins	8
Serial Data Pins	9
GPS Breakouts	10
Coin Battery	11
Reset Button	12
Antennas	13
Basic RX-TX Test	14
Arduino Library	18
Parsed Data	18
Battery Backup	21
Antenna Options	23
Resources	25
Datasheets	25
More reading:	25
Adafruit GPS Library for Arduino	25
EPO files for AGPS use	25
F.A.Q.	26
Downloads	28
Datasheets & Files	28
Schematic	28
Fabrication Print	29

Overview

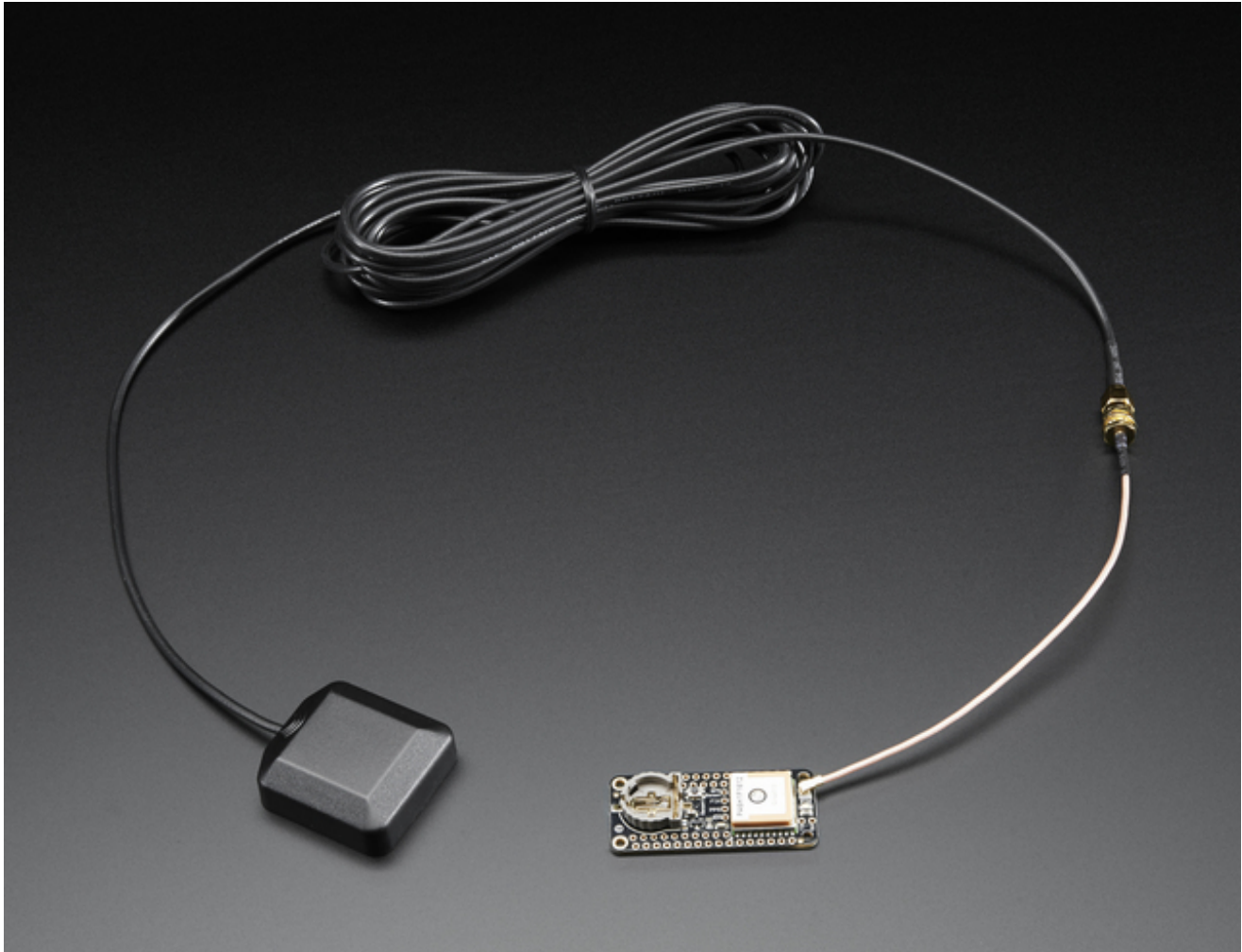


Give your Feather a sense of place, with an Ultimate GPS FeatherWing. This FeatherWing plugs right into your Feather board and gives it a precise, sensitive, and low power GPS module for location identification anywhere in the world. As a bonus, the GPS can also keep track of time once it is synced with the satellites.

- -165 dBm sensitivity, 10 Hz updates, 66 channels
- 20mA current draw
- RTC with coin battery backup
- Built-in datalogging
- PPS (pulse per second) output on fix
- Internal patch antenna + u.FL connector for external active antenna
- Fix status LED

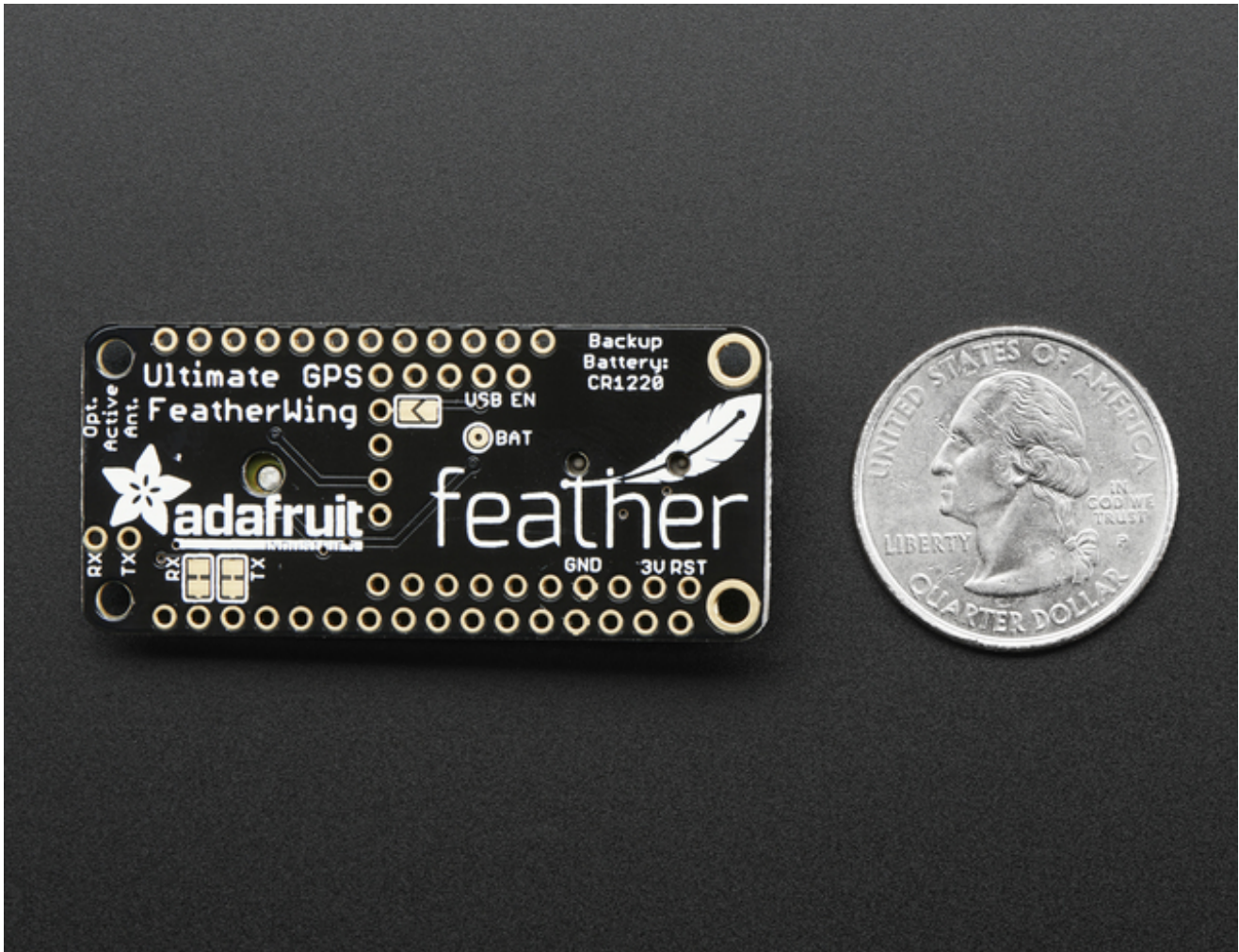


The Wing is built around the MTK3339 chipset, a no-nonsense, high-quality GPS module that can track up to 22 satellites on 66 channels, has an excellent high-sensitivity receiver (-165 dB tracking!), and a built in antenna. It can do up to 10 location updates a second for high speed, high sensitivity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



The MTK3339-based module has external antenna functionality. 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>)

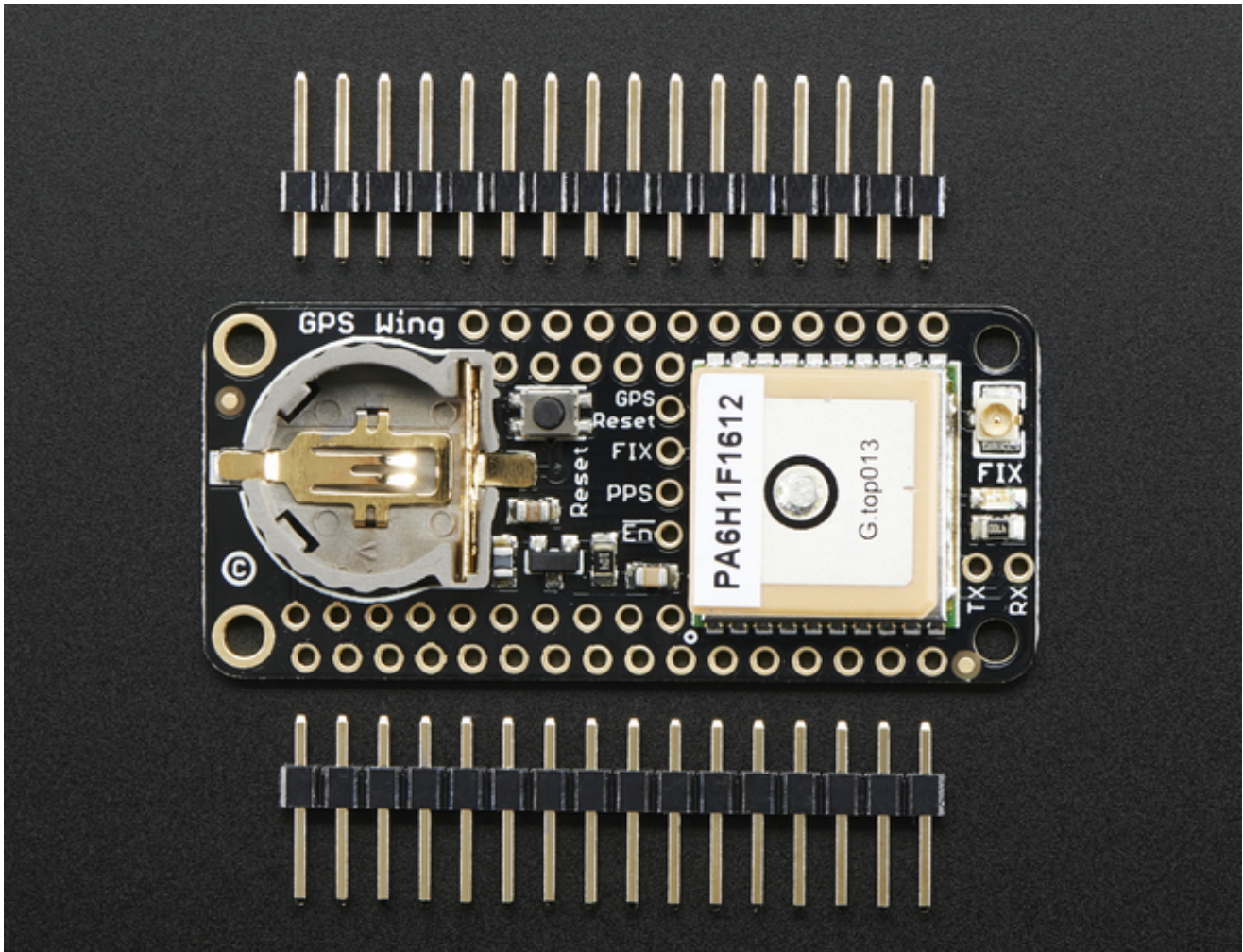
We added some extra goodies to make this GPS FeatherWing better than a breakout: a FET-driven ENABLE pin so you can turn off the module using any microcontroller pin for low power use, a CR1220 coin cell holder to keep the RTC running and allow warm starts and a tiny bright red LED. The LED blinks at about 1Hz while it's searching for satellites and blinks once every 15 seconds when a fix is found to conserve power.



Module specs:

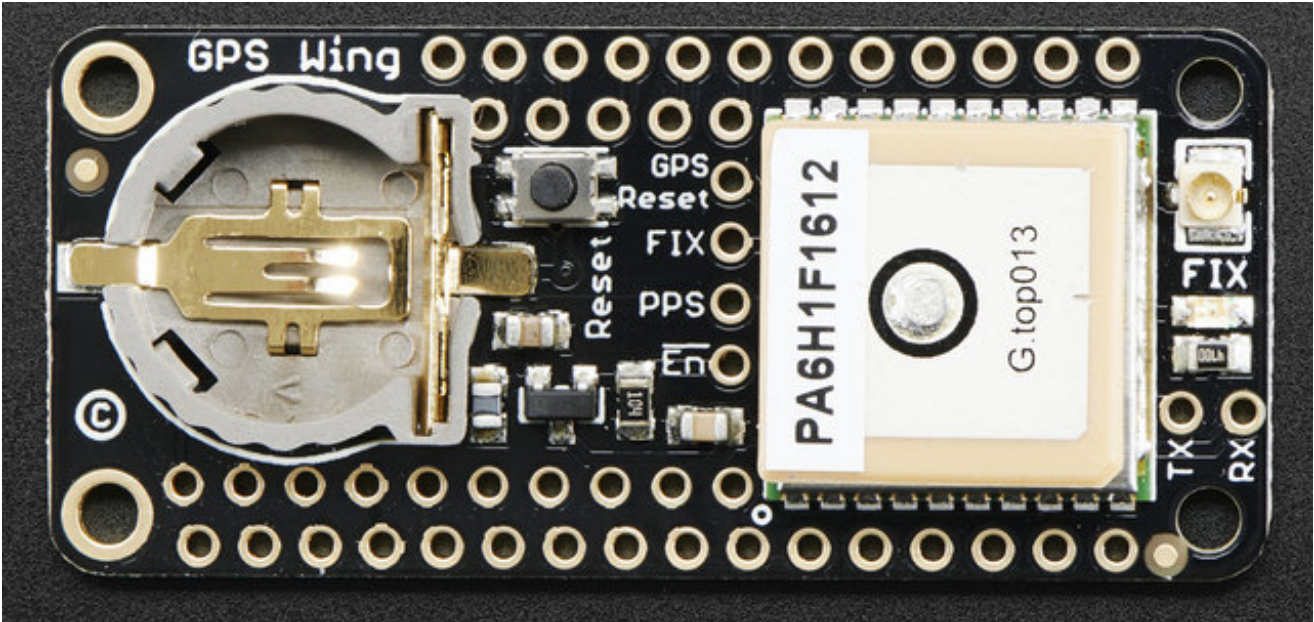
- Satellites: 22 tracking, 66 searching
- Patch Antenna Size: 15mm x 15mm x 4mm
- Update rate: 1 to 10 Hz
- Position Accuracy: 1.8 meters
- Velocity Accuracy: 0.1 meters/s
- Warm/cold start: 34 seconds
- Acquisition sensitivity: -145 dBm
- Tracking sensitivity: -165 dBm
- Maximum Velocity: 515m/s
- Vin 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

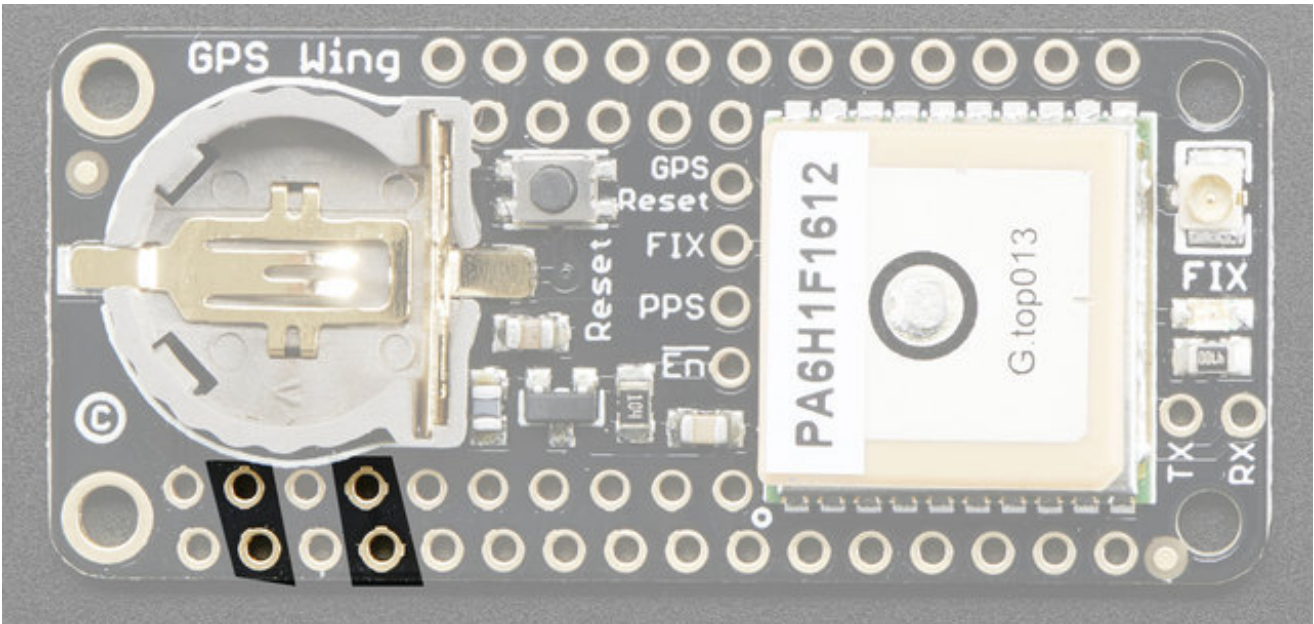


Comes as a fully assembled FeatherWing with GPS module and a coin cell holder. However, the 12mm coin cell is **not included** - it isn't required but if you would like a battery backup, a CR1220 coin battery must be purchased separately. We also toss in some header so you can solder it in and plug into your Feather

Pinouts

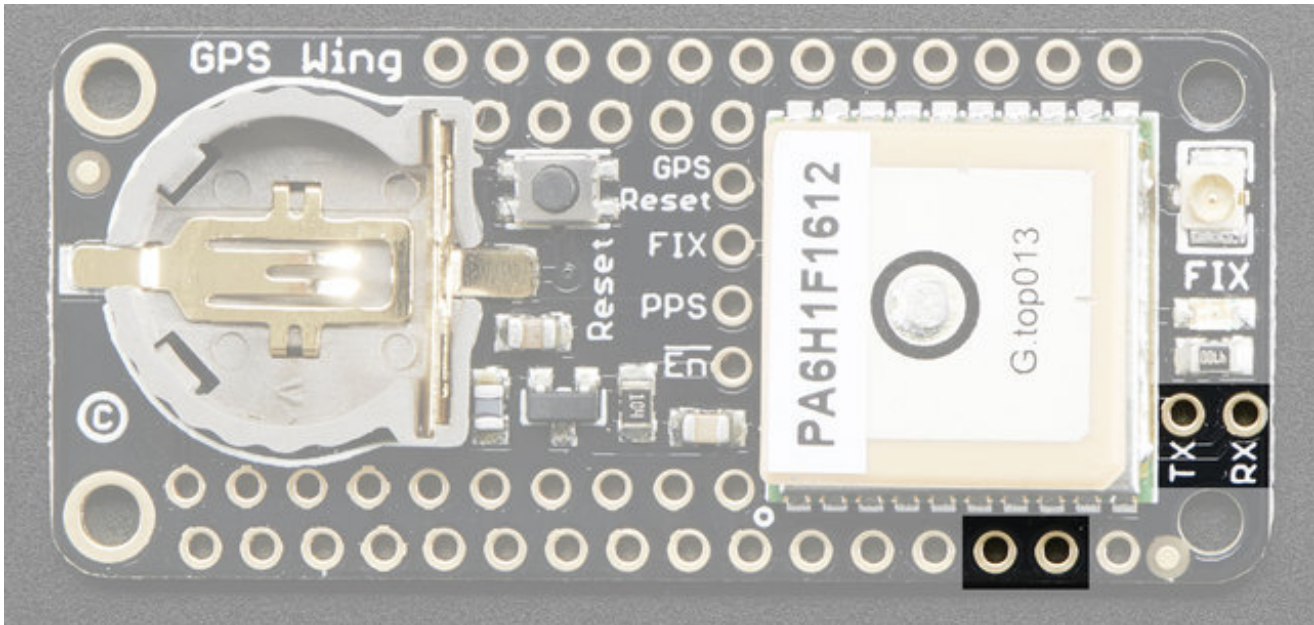


Power Pins



The GPS module runs from +3.3V power and uses 3V logic. the GPS is powered directly from the **3V** and **GND** pins on the bottom left of the Feather. Each Feather has a regulator to provide clean 3V power from USB or battery power.

Serial Data Pins

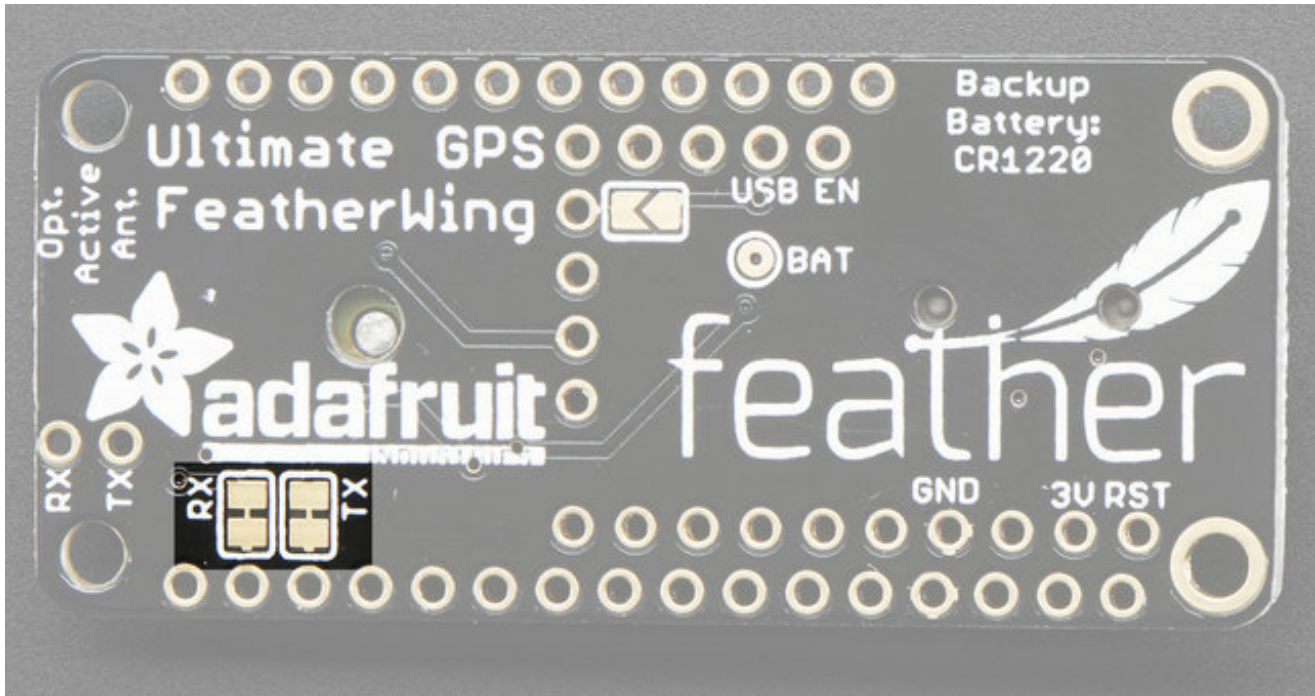


The GPS module, like pretty much all GPS's, communicates over UART serial. It sends ASCII NMEA sentences from the GPS TX pin to the microcontroller RX pin and can be controlled to change its data output from the GPS RX pin. Logic level is 3.3V for both.

The baud rate by default is 9600 baud, but you can configure the module to use different baud rate if desired

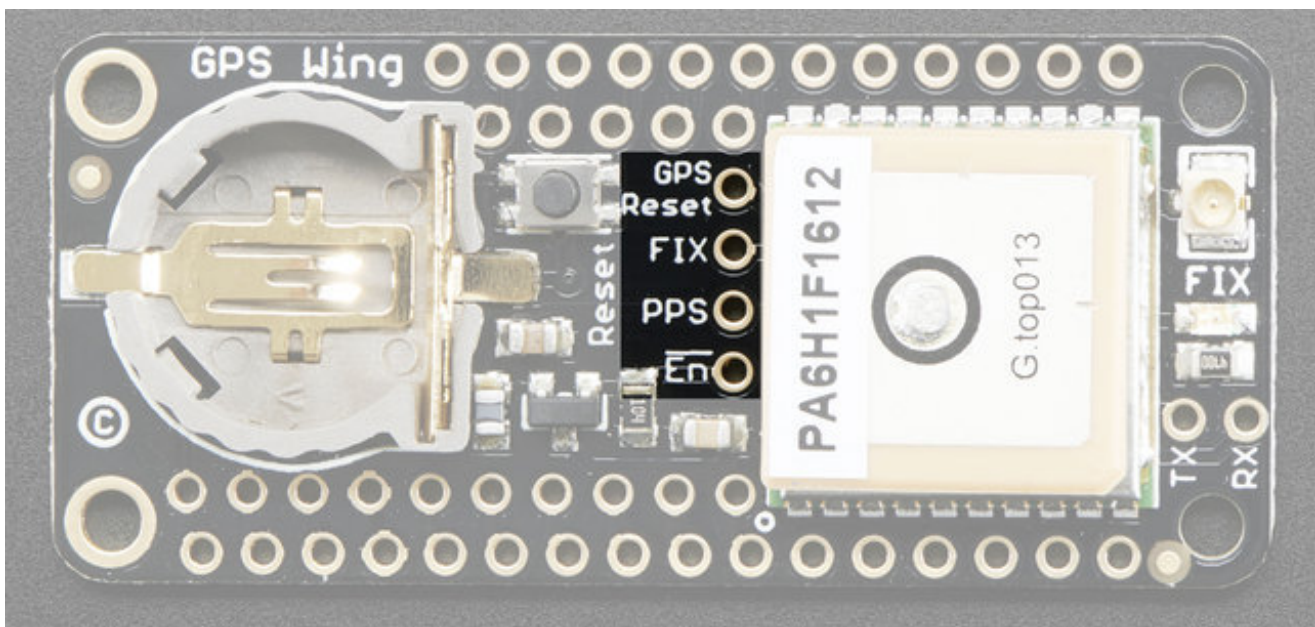
The GPS is wired directly to the Serial pins at the bottom right of the Feather. **Note that on the ESP8266 Feather this is used for bootloading so the GPS module is not recommended for use with the ESP8266!** The other Feathers use USB for bootloading, so the hardware Serial port is available.

If you need to connect to a different set of pins, you can cut the RX and TX jumpers on the bottom of the board



And then wire your desired pins to the RX and TX breakouts right next to the **FIX** LED

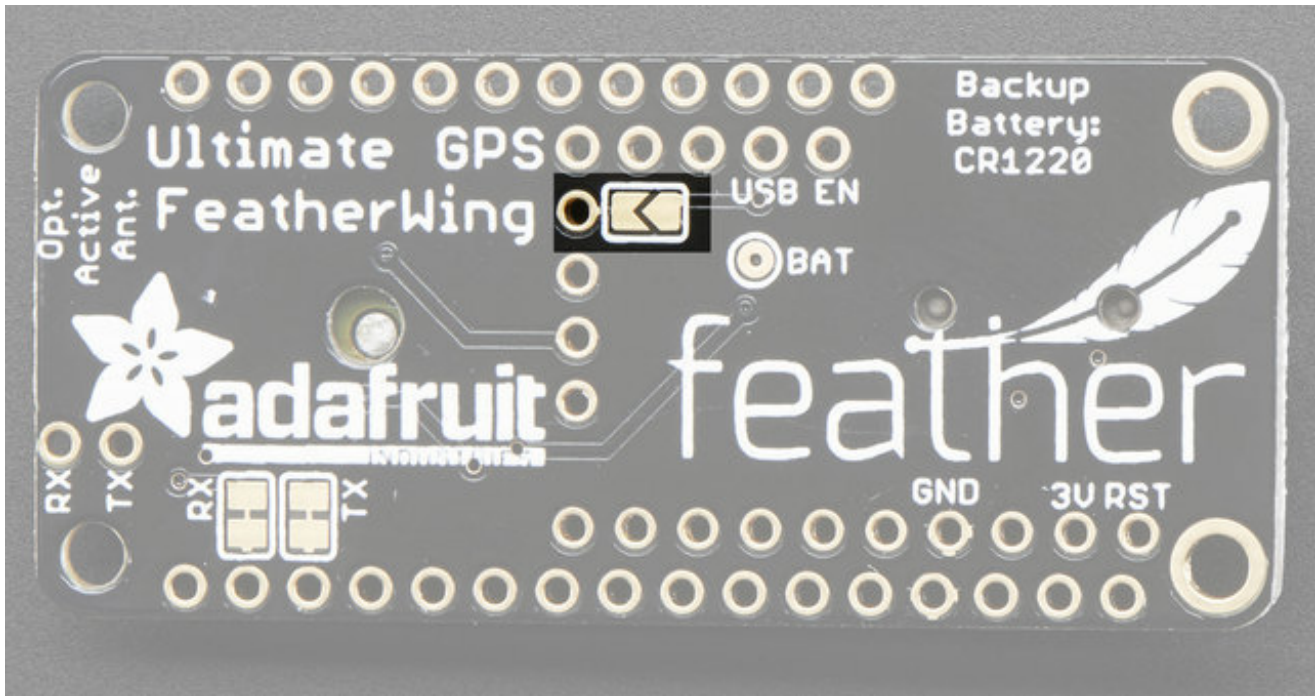
GPS Breakouts



The GPS module has some more pins you can use:

GPS Reset - you can use this to manually reset the GPS module if you set the baud rate to something inscrutable or you just want to 'kick it'. Reset by pulling low. Note that pulling reset low does not put the GPS into a low power state, you'll want to disable it instead (see

EN). There's a jumper on the back if you want to connect the GPS reset to the microcontroller reset line

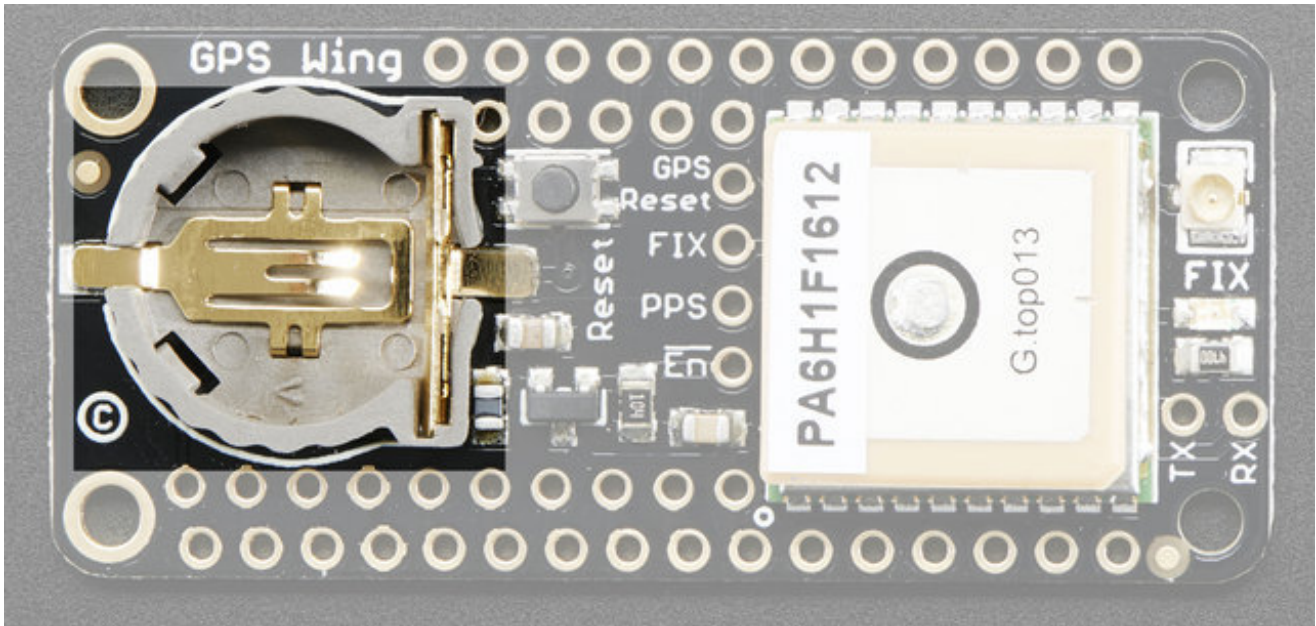


FIX is an output pin - it is the same pin as the one that drives the red FIX LED. When there is no fix, the FIX pin is going to pulse up and down once every second. When there is a fix, the pin is low (0V) for most of the time, once every 15 seconds it will pulse high for 200 milliseconds

PPS is a "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

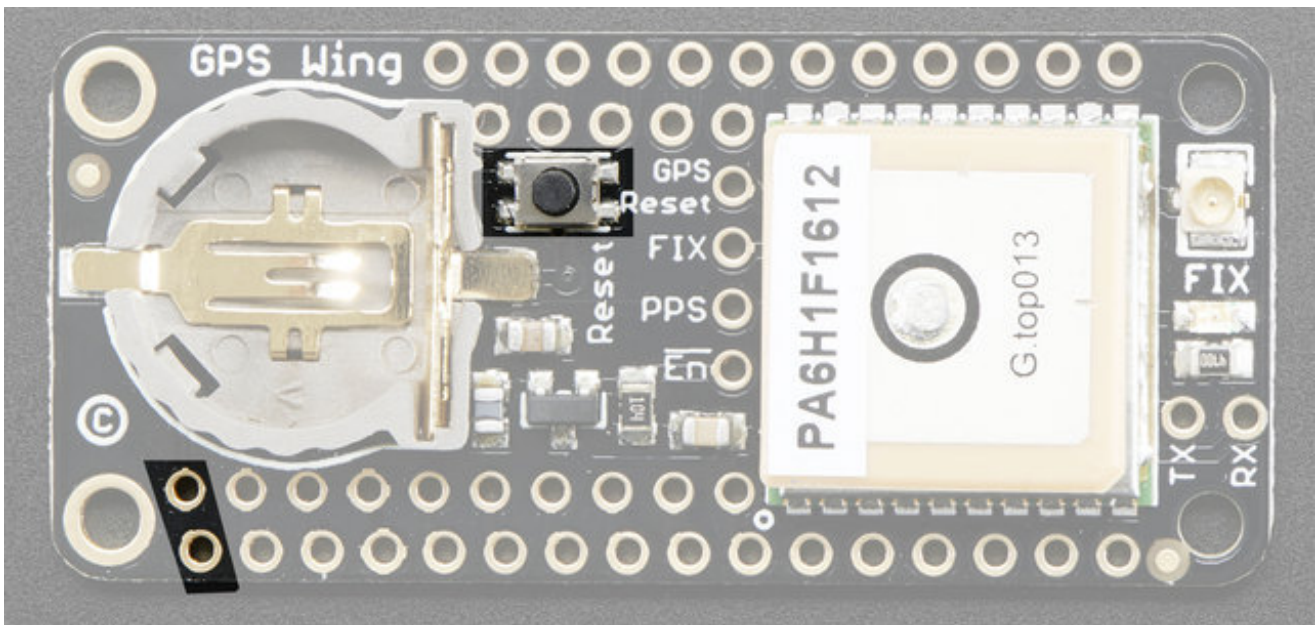
EN is a true 'power disable' control line you can use to completely cut power to the GPS module. This is good if you need to run at ultra-low-power modes. By default this is pulled low (enabled). So pull high to disable the GPS.

Coin Battery



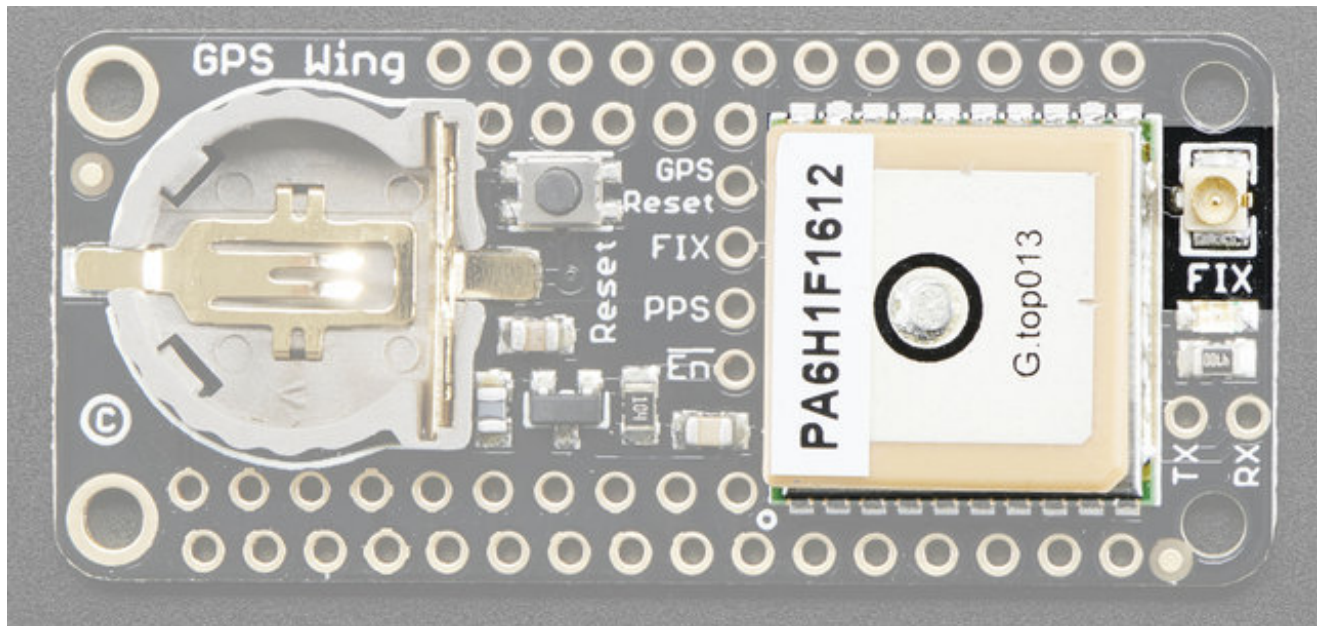
The GPS does not *require* but does benefit by having a coin cell backup battery. You can install any "CR1220" sized battery into the holder. The GPS will automatically detect the battery and use it as a backup for the ephemeris memory

Reset Button



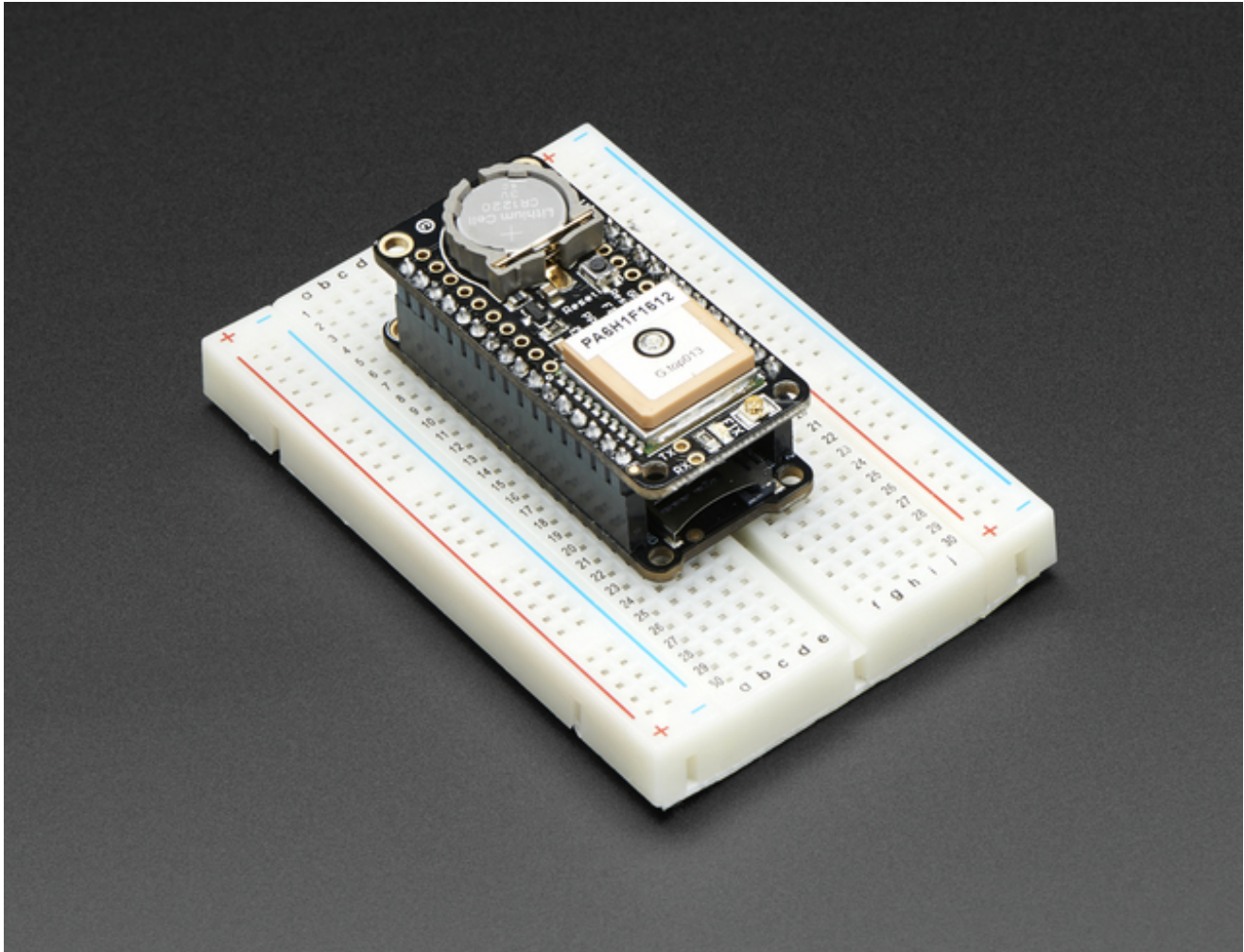
There is a small button that will connect the *microcontroller* RESET pin to ground for resetting it. Handy to restart the Feather. Note that this is not connected to GPS reset unless you short the jumper on the back!

Antennas



You have two options for an antenna. The GPS module has an antenna on it already, that's the square tan ceramic thing on the left side. If you plug an *active GPS antenna* into the uFL connector on the very right, the module will automatically switch over to the external antenna.

Basic RX-TX Test



The first, and simplest test, is just to echo data back and forth using the Feather's USB connection to read the GPS data out. The nice thing about this demo is no library is required.

This doesn't work with the ESP8266 or nRF52 Feather!

Upload the following to your Feather, it will work with the Feather 32u4, M0, WICED, etc. Anything that does not have the RX/TX pins used to communicate.

```
// test a passthru between USB and hardware serial

void setup() {
  Serial.println("GPS echo test");
  Serial.begin(9600);
}
```



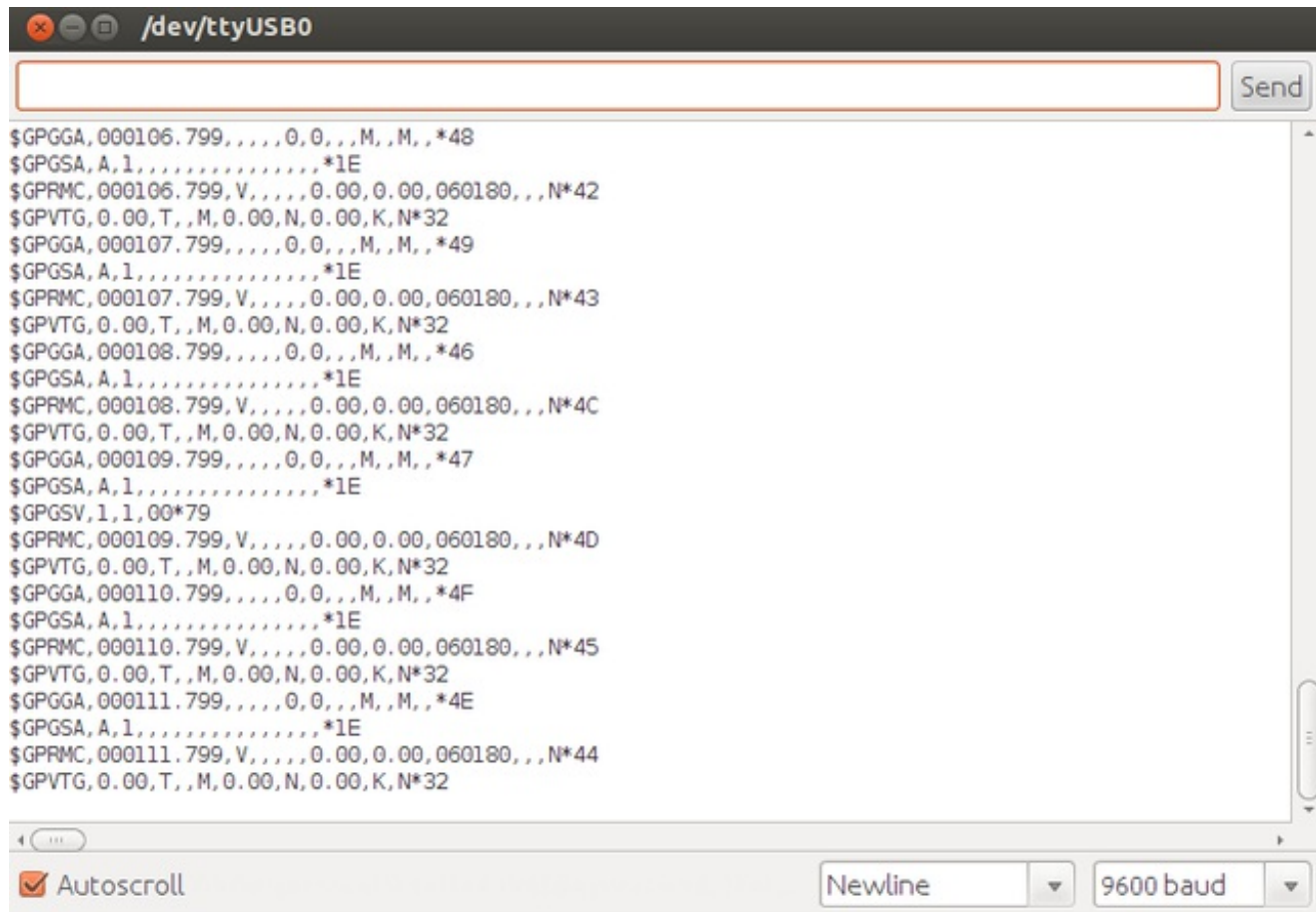
```

Serial1.begin(9600); // default NMEA GPS baud
}

void loop() {
  if (Serial.available()) {
    char c = Serial.read();
    Serial1.write(c);
  }
  if (Serial1.available()) {
    char c = Serial1.read();
    Serial.write(c);
  }
}
}

```

Now 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:



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** (**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

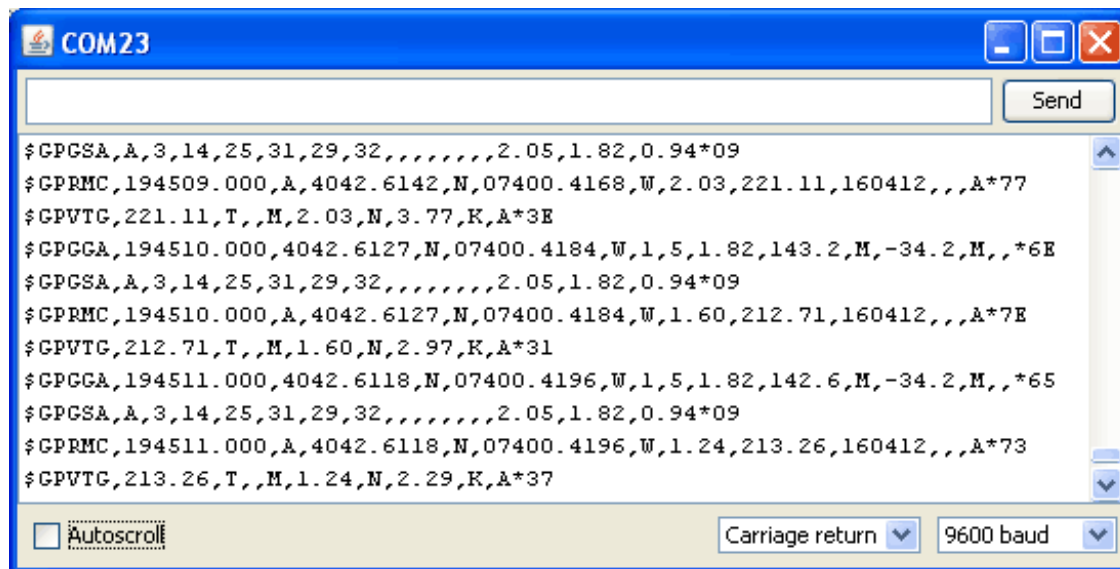
onto the satellite data and received enough data to determine the location (2D fix) or location+altitude (3D fix).

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

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) 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:



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](http://adafru.it/aMI) (<http://adafru.it/aMI>) . 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.



Arduino Library

Once you've gotten the GPS module tested with direct rx-tx, we can go forward and do some more advanced parsing.

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. [To download it, visit the GitHub repository \(http://adafru.it/aMm\)](http://adafru.it/aMm) or just click below

[Download Adafruit GPS Library](http://adafru.it/emg)
<http://adafru.it/emg>

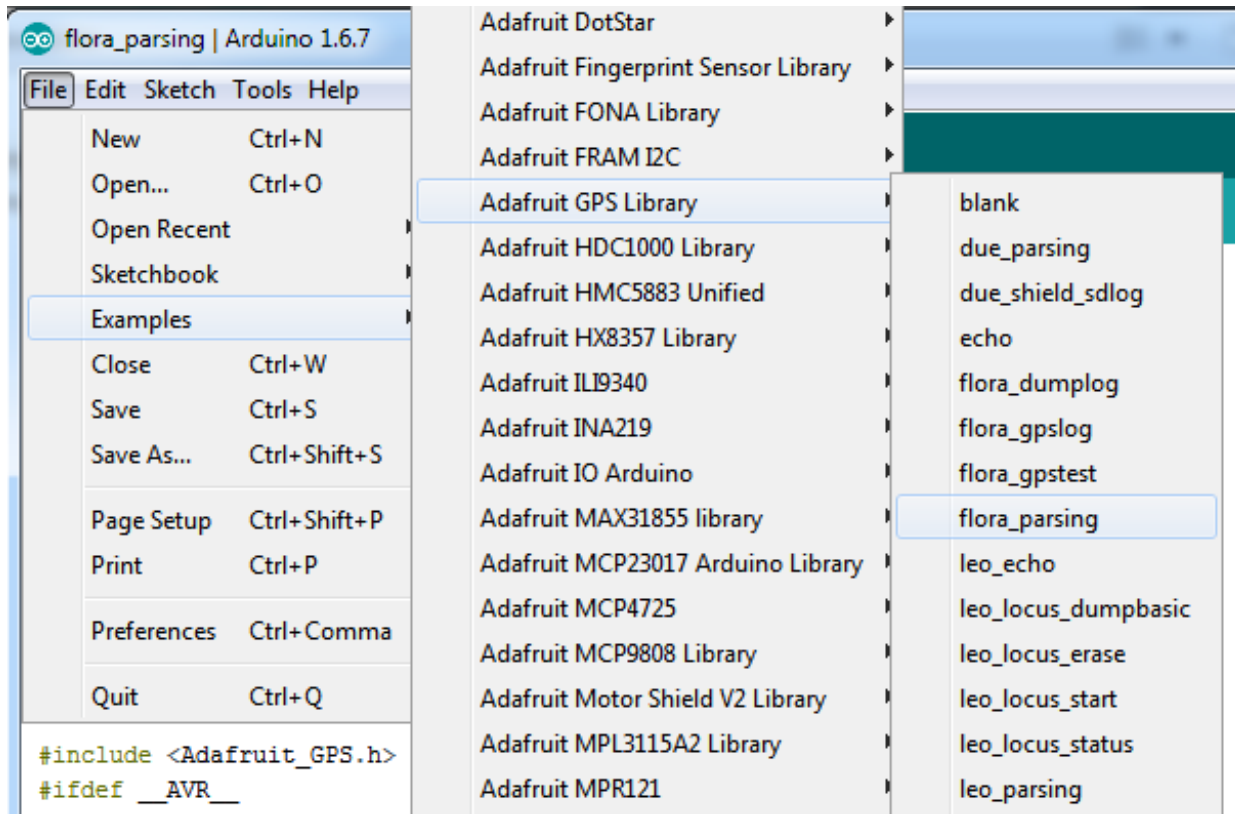
Rename the uncompressed folder **Adafruit_GPS**. Check that the **Adafruit_GPS** folder contains **Adafruit_GPS.cpp** and **Adafruit_GPS.h**

Move **Adafruit_GPS** to your Arduino/Libraries folder and restart the Arduino IDE. Library installation is a frequent stumbling block...if you need assistance, our [All About Arduino Libraries \(http://adafru.it/dit\)](http://adafru.it/dit) guide spells it out in detail!

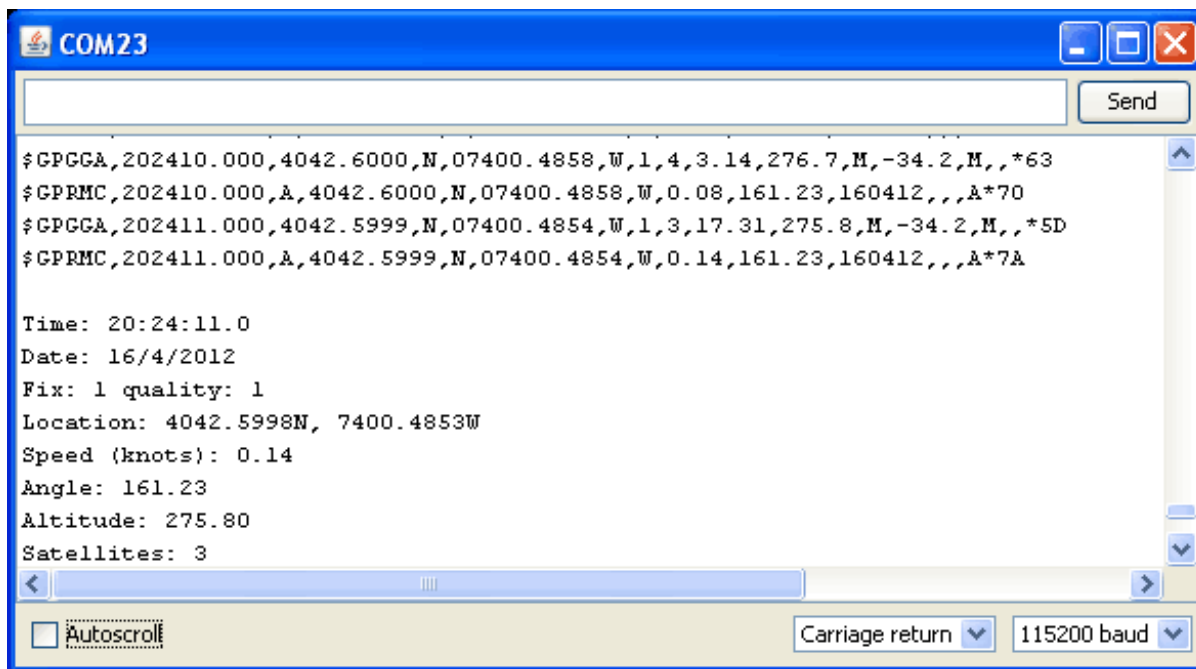
Parsed Data

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→flora_parsing** sketch and upload it to the Arduino.



Then open up the serial monitor.



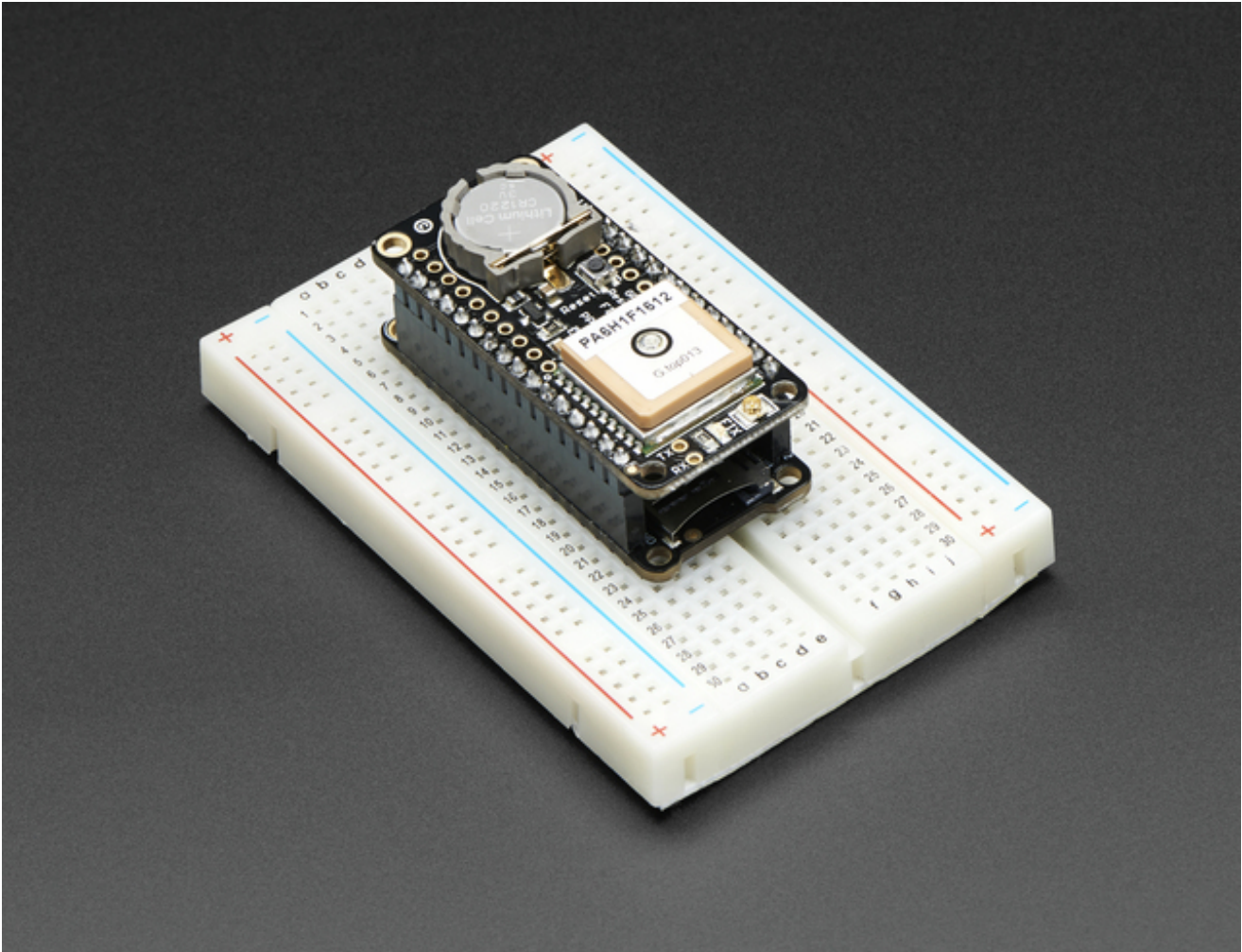
In this sketch, we call **GPS.read()** constantly in the main loop (if you can, get this to run once a millisecond in an interrupt). Then in the main loop we can ask if a new chunk of data has been received by calling **GPS.newNMEAreceived()**, if this returns **true** then we can ask the library to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

Battery Backup

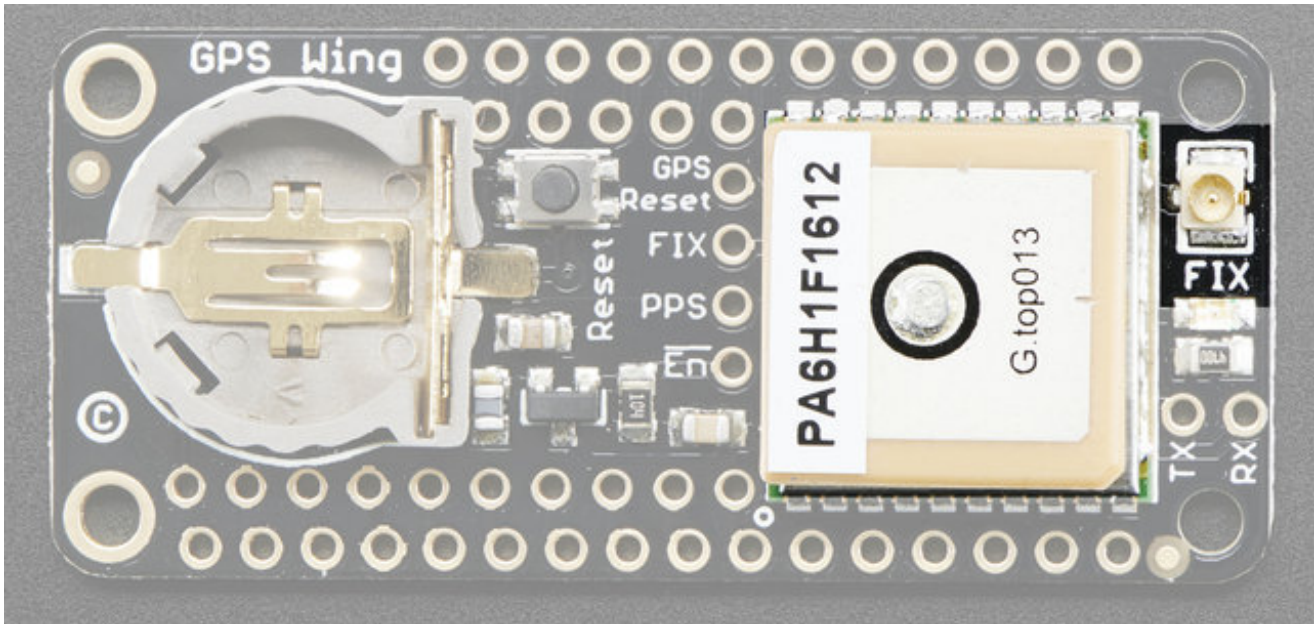


The GPS has a built in real time clock, which can keep track of time even when it power is lost and it doesn't have a fix yet. It can also help reduce fix times, if you expect to have a flakey power connection (say you're using solar or similar). To use the RTC, we need to install a battery. We provide 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 μA (0.007 mA) so a CR1220 will last $40\text{mAh} / 0.007\text{mA} = 5,714$ 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

Antenna Options



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://adafruit.it/960) (<http://adafruit.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://adafruit.it/960) (<http://adafruit.it/960>)

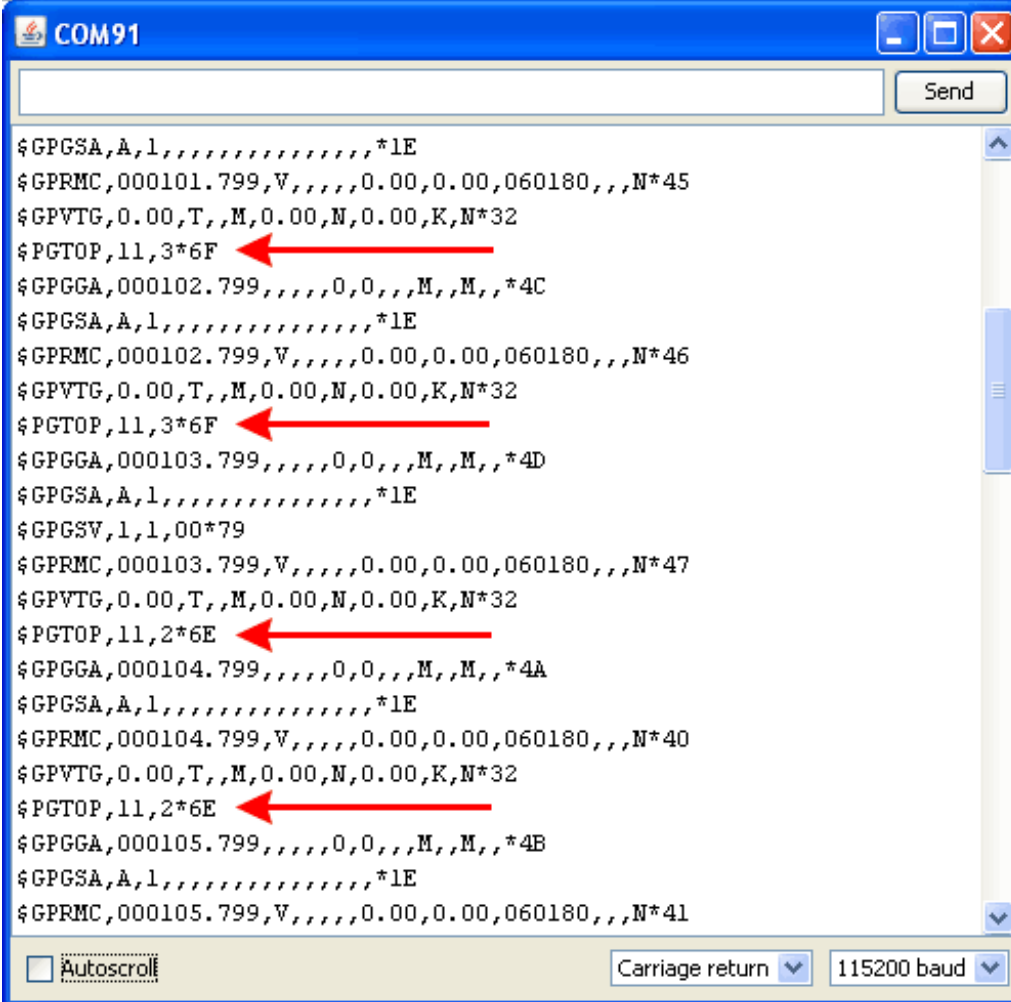
Most GPS antennas use SMA connectors, which are popular and easy to use. However, an SMA connector would be fairly big on the GPS breakout so we went with a uFL connector - which is lightweight, small and easy to manufacture. If you don't need an external antenna it wont add significant weight or space but [its easy to attach a uFL->SMA adapter cable](http://adafruit.it/851) (<http://adafruit.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** 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 adding a **gps.sendCommand(PGCMD_ANTENNA)** around the same time you set the update rate/sentence output.



```
COM91
$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
```

Autoscroll Carriage return 115200 baud



Resources

Datasheets

- [MTK3329/MTK3339 command set sheet \(http://adafru.it/e7A\)](http://adafru.it/e7A) for changing the fix data rate, baud rate, sentence outputs, etc!
- [PMTK 'complete' data \(http://adafru.it/uoe\)](http://adafru.it/uoe) sheet (like the above but with even more commands)
- [Datasheet for the PA6B \(MTK3329\) GPS module itself \(http://adafru.it/aMo\)](http://adafru.it/aMo)
- [Datasheet for the PA6C \(MTK3339\) GPS module itself \(http://adafru.it/aMp\)](http://adafru.it/aMp)
- [Datasheet for the PA6H \(MTK3339\) GPS module itself \(http://adafru.it/aPO\)](http://adafru.it/aPO)
- [MT3339 GPS PC Tool \(windows only\) \(http://adafru.it/uof\)](http://adafru.it/uof) and the [PC Tool manual \(http://adafru.it/uoA\)](http://adafru.it/uoA)
- [Sample code and spec sheet for the LOCUS built-in logger \(http://adafru.it/aTi\)](http://adafru.it/aTi)
- [LOCUS \(built-in-datalogging system\) user guide \(http://adafru.it/uoc\)](http://adafru.it/uoc)
- [Mini GPS tool \(windows only\) \(http://adafru.it/aMs\)](http://adafru.it/aMs)

More reading:

- [Trimble's GPS tutorial \(http://adafru.it/emh\)](http://adafru.it/emh)
- [Garmin's GPS tutorial \(http://adafru.it/aMv\)](http://adafru.it/aMv)

Adafruit GPS Library for Arduino

<https://github.com/adafruit/Adafruit-GPS-Library/> (<http://adafru.it/emi>)

EPO files for AGPS use

[Data format for EPO files \(http://adafru.it/uoB\)](http://adafru.it/uoB)

[MTK_EPO_Nov_12_2014.zip](http://adafru.it/eb1)

<http://adafru.it/eb1>



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 than 5223 then it's 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

OK I want the latest firmware!

[Here is the binary of the 5632 firmware \(http://adafru.it/dR5\)](http://adafru.it/dR5), you can [use this tool to upload it using an FTDI or USB-TTL cable \(or direct wiring with FTDI\) \(http://adafru.it/uoF\)](http://adafru.it/uoF). 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!

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 accessible 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!



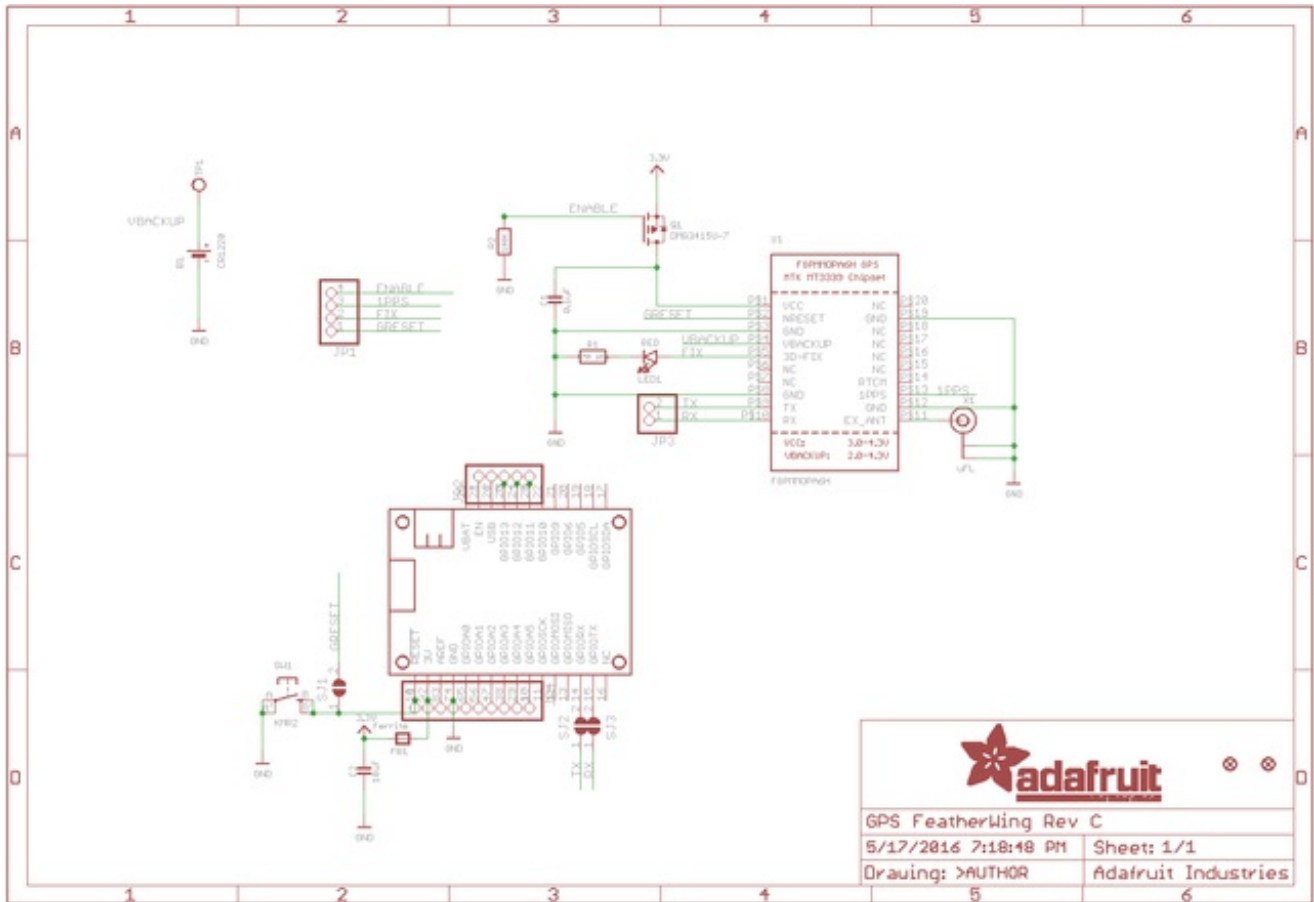
Downloads

Datasheets & Files

- [EagleCAD PCB Files on GitHub](http://adafru.it/nCQ) (<http://adafru.it/nCQ>)
- [Fritzing object in Adafruit Fritzing library](http://adafru.it/aP3) (<http://adafru.it/aP3>)
- [Adafruit GPS Arduino Library](http://adafru.it/nCR) (<http://adafru.it/nCR>)
- [MTK3329/MTK3339 command set sheet](http://adafru.it/qif) (<http://adafru.it/qif>) for changing the fix data rate, baud rate, sentence outputs, etc!
- [LOCUS \(built-in-datalogging system\) user guide](http://adafru.it/uoC) (<http://adafru.it/uoC>)
- [Datasheet for the PA6H \(MTK3339\) GPS module itself](http://adafru.it/ria) (<http://adafru.it/ria>)
- [MT3339 GPS PC Tool \(windows only\)](http://adafru.it/uoD) (<http://adafru.it/uoD>) and the [PC Tool manual](http://adafru.it/uoE) (<http://adafru.it/uoE>)
- [Mini GPS tool \(windows only\)](http://adafru.it/rid) (<http://adafru.it/rid>)

Schematic

Click to embiggen



Fabrication Print

Dimensions in inches

