



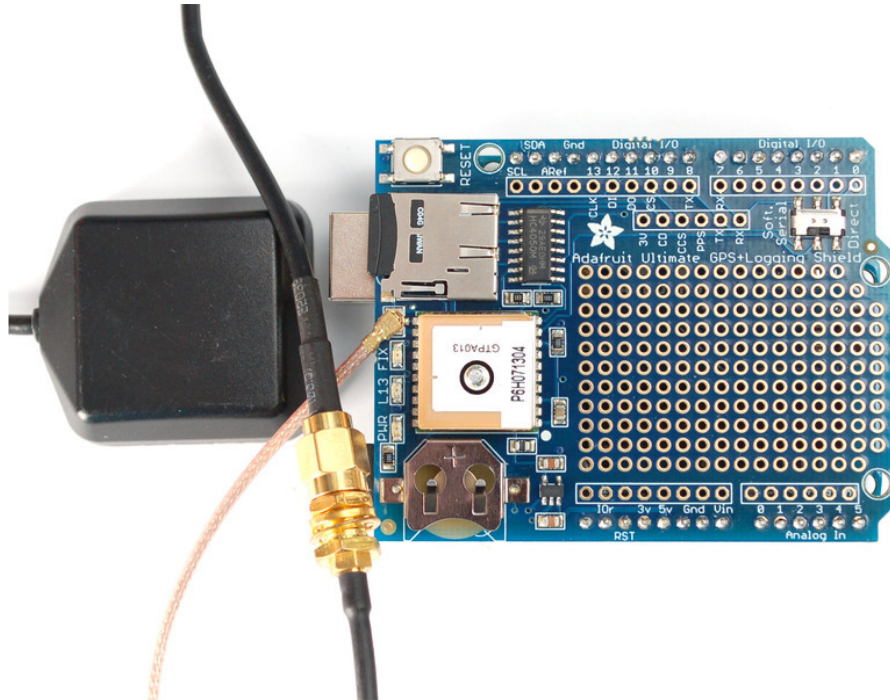
## Guide Contents

Guide Contents	2
Overview	3
Shield Headers	5
Assembly with male headers	5
Assembly with stacking headers	7
Direct Connect	9
Direct Connection Using the Switch (Uno/Mega)	9
Direct Connection with Jumpers on Leonardo	10
Testing Direct Connection	11
Sending NMEA Commands via Direct Connect	13
Soft Serial Connect	15
Hardware Serial Connect	17
Soft Serial	17
Direct Connect	17
Hardware Serial	18
Metro M0/M4 Example	19
Parsing Data	21
If you're using an Adafruit Metro 328P or Arduino Uno (or older)	21
If you're using an Arduino Leonardo..	21
SD Logging	23
SD Card Library Update	23
Built In Logging	27
Built In Logging	27
Logging Status	28
Downloading Data	28
Using the GPS Tool	29
LOCUS Parser	30
Antenna, Battery and More!	31
LEDs	31
Antennas	31
RTC Backup Battery	33
Breakout and Proto-area	34
Downloads	35
Schematic	35
F.A.Q.	36
Can the Ultimate GPS be used for High Altitude? How can I know?	36
Is the Ultimate GPS affected by the 2019 Week Rollover issue?	37
OK I want the latest firmware!	38
I've adapted the example code and my GPS NMEA sentences are all garbled and incomplete!	39
My GPS is giving me data but the location is wrong!	40
How come I can't get the GPS to output at 10Hz?	41
How come I can't set the RTC with the Adafruit RTC library?	42
Do all GPS modules emit PPS pulses at the same time?	43
Why am I not seeing anything on the PPS pin?	44
How can I read the PPS signal on the Ultimate GPS USB?	45



Simply connect an [external active GPS antenna](http://adafru.it/960) (<http://adafru.it/960>) via a [uFL/SMA cable](http://adafru.it/851) (<http://adafru.it/851>) to the shield and the module will automatically switch over to use the antenna. You can then place the antenna wherever you wish.

We think this is the Ultimate GPS shield and we also think you'll agree!



# Shield Headers

The Ultimate GPS Logger shield comes tested assembled with a GPS unit and microSD socket already on it, but you'll still need need to put headers on so you can plug it into an Arduino

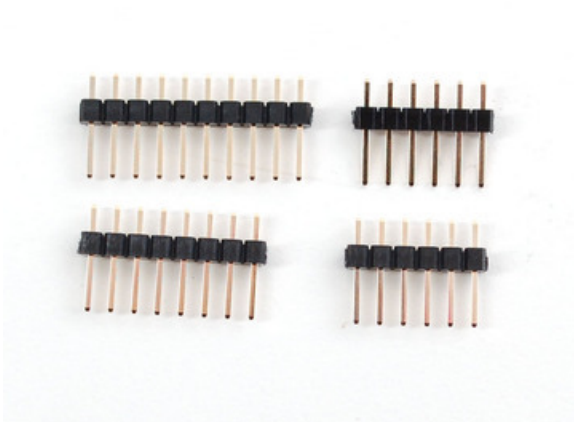
We don't pre-assemble the headers on because there's **two** options! You can either use plain 0.1" male headers (included with the shield) or [Arduino Shield Stacking headers](http://adafru.it/85) (<http://adafru.it/85>).

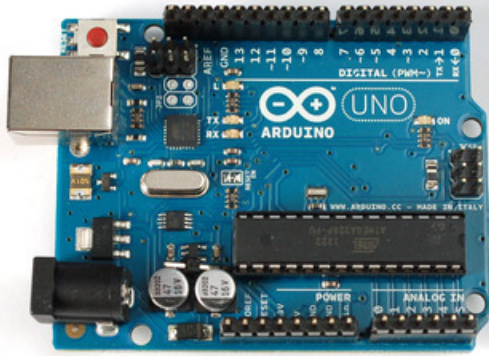
## Assembly with male headers

Most people will be happy with assembling he shield with male headers. The nice thing about using these is they don't add anything to the height of the project, and they make a nice solid connection. However, you won't be able to stack another shield on top. Trade offs!

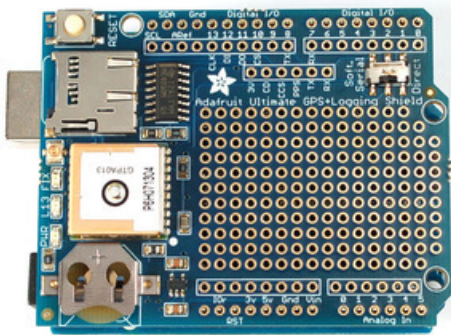


If you want to go with male headers, break off the 36-pin long stick into either 2 x 8pin and 2 x 6pin *or* 1 x 10pin, 2 x 8pin and 1 x 6pin (depending on how many headers your Arduino has and which version it is)

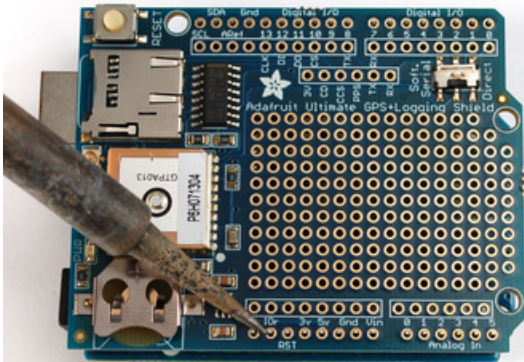


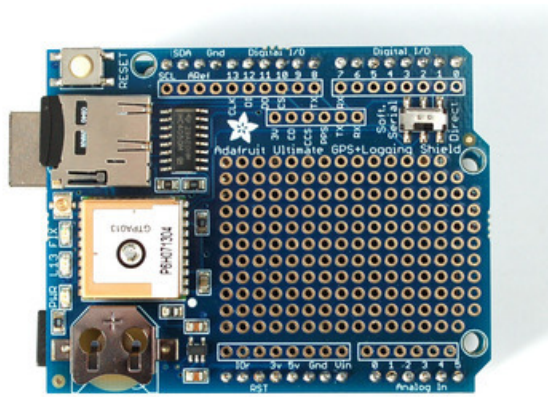


Place the header onto your Arduino, with the long pins down.



Place the shield on top so the short ends stick out thru the PCB, solder all the pins





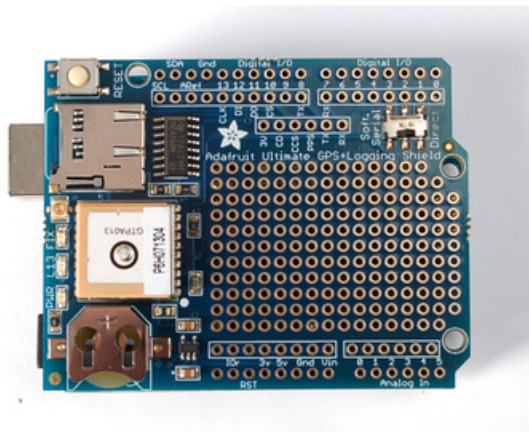
That's it! Now you can continue and test the shield

## Assembly with stacking headers

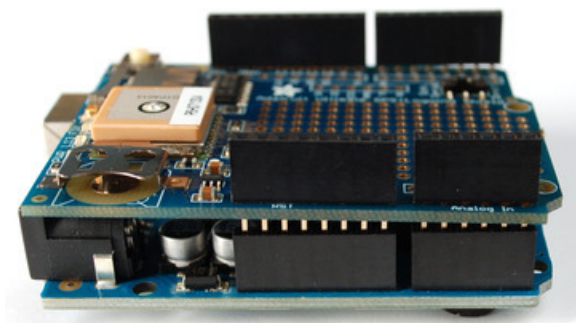
If you want to attach another shield on top, you can use stacking headers. These headers provide a 'pass-thru' connection so multiple shields can be attached. However, if you put a shield on top, its likely it will cover the GPS patch antenna so you may need to get an [external GPS antenna \(http://adafru.it/960\)](http://adafru.it/960). Stacking headers also make the project taller (harder to fit into a small box) and the headers are thinner so connection is not as strong (so if the project is going to be shaken around a lot it may not make good connections during harsh movement)

If you want to go with stacking headers, place the the shield PCB on top of the Arduino and fit the long pins through the pads until they slot into the Arduino. Depending on which version/type of Arduino you may need to use 2 x 8pin and 2 x 6pin *or* 1 x 10pin, 2 x 8pin and 1 x 6pin.

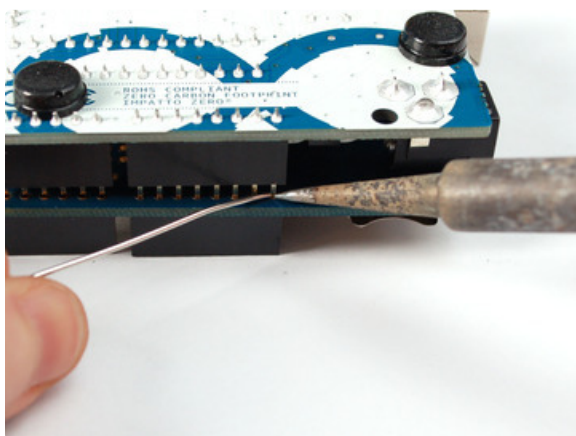
Turn the Arduino upside down so its resting on the long parts of the stacky headers and solder them from below



If you want to go with stacking headers, place the the shield PCB on top of the Arduino



Fit the long pins through the pads until they slot into the Arduino. Depending on which version/type of Arduino you may need to use 2 x 8pin and 2 x 6pin *or* 1 x 10pin, 2 x 8pin and 1 x 6pin.



Turn the Arduino upside down so its resting on the long parts of the stacky headers and solder them from below.

You may want to pull the headers out a little to get them to sit flat against the shield PCB

Solder all the pins!



# Direct Connect

Direct connection is a funky trick we use to connect the output of the GPS serial TTL UART directly to the usb-serial converter chip on an Arduino. This takes the Arduino out of the picture and can make it easy if you want to experiment sending commands directly, or using the Windows software (the Arduino would act like a USB->UART bridge)

## Direct Connection Using the Switch (Uno/Mega)

Direct Connection by flipping the switch on this shield only works if you're using an Uno/Duemilanove/Diecimila/compatible OR a Mega (any type). It will not work with a Leonardo (the Leonardo does not have a USB/Uart chip) or Due (haven't tested, but assume it doesn't)

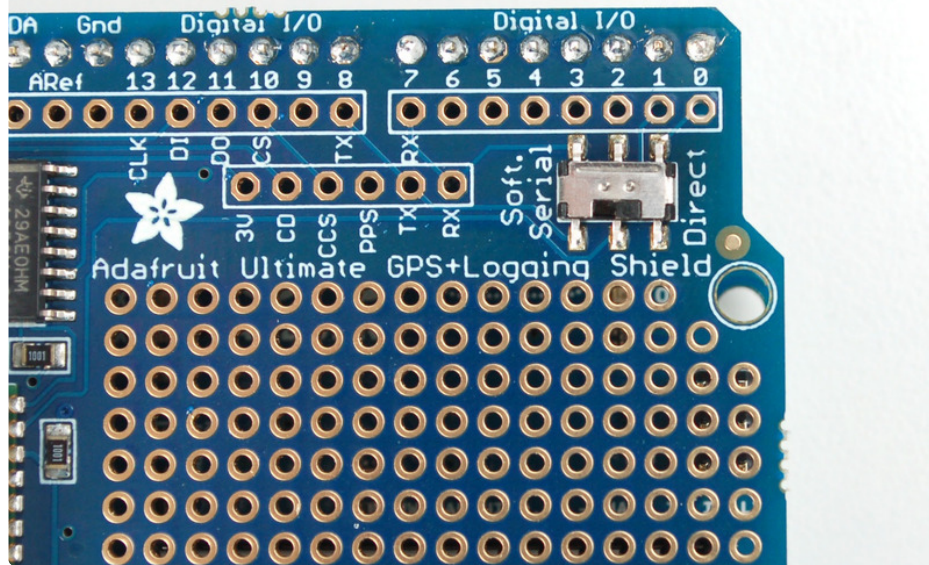
First, load a 'blank' sketch into the Arduino:

```
// this sketch will allow you to bypass the Atmega chip
// and connect the 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() {}
```

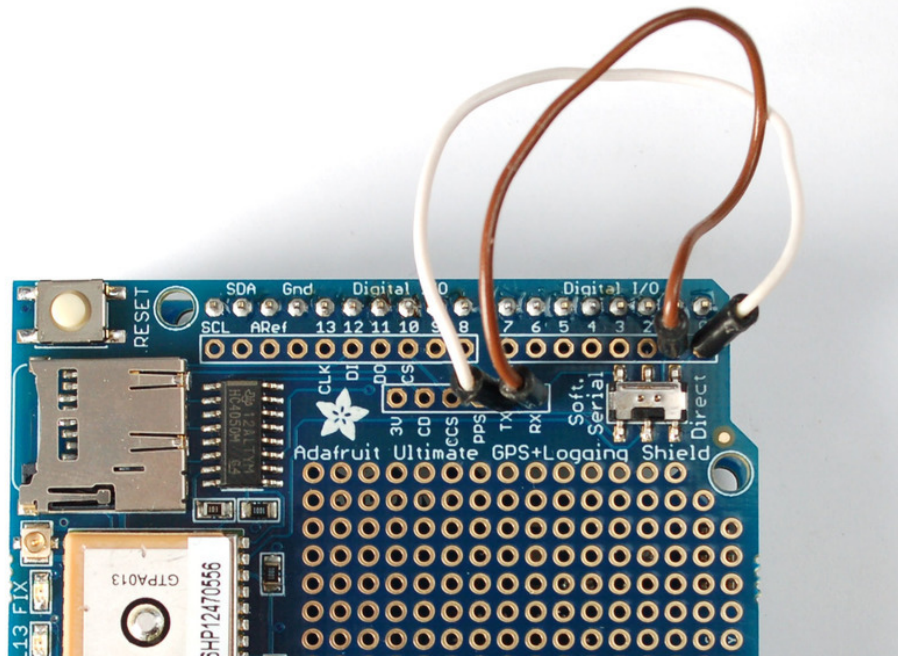
This will free up the converter so you can directly wire and bypass the Arduino chip. Once you've uploaded this sketch, flip the switch on the shield to **Direct**



## Direct Connection with Jumpers on Leonardo

If you have a Leonardo we have to do a funky trick where we swap the Direct connect wires (because the processor chip acts like the USB/Serial converter rather than having a separate chip. The upshot is you'll need two wires. For basic testing as long as the wires touch the two sets of pads, you'll be able to continue with this very basic test. We don't suggest soldering them yet in since as long as the GPS works, you can go forward and use software serial if you prefer

Select Software Serial on the switch. Connect a wire from the **TX** pad to digital **0** and a wire from the **RX** pad to digital **1**.



Finally upload the `Adafruit_GPS->leo_echo` sketch to the Leonardo, this will shuffle data from the GPS to the USB port

## Testing Direct Connection

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:

A screenshot of the Arduino IDE Serial Monitor window. The window title is "COM7". The text area displays several lines of raw NMEA sentences, including \$GPGSA, \$GPRMC, \$GPVTG, and \$GPGGA. The baud rate is set to 9600. The "Autoscroll" checkbox is checked, and the "Carriage return" dropdown is set to "None".

```
COM7
$GPGSA,A,1,,,,,,,,,,,,,*1E
$GPRMC,002312.317,V,,,,,0.00,0.00,060180,,,N*45
$GPVTG,0.00,T,,M,0.00,N,0.00,K,N*32
$GPGGA,002313.317,,,,,0,0,,,M,,M,,*4E
$GPGSA,A,1,,,,,,,,,,,,,*1E
$GPRMC,002313.317,V,,,,,0.00,0.00,060180,,,N*44
$GPVTG,0.00,T,,M,0.00,N,0.00,K,N*32
$GPGGA,002314.317,,,,,0,0,,,M,,M,,*49
$GPGSA,A,1,,,,,,,,,,,,,*1E
$GPGSV,1,1,00*79
$GPRMC,002314.317,V,,,,,0.00,0.00,060180,,,N*43
$GPVTG,0.00,T,,M,0.00,N,0.00,K,N*32
$GPGGA,002315.317,,,,,0,0,,,M,,M,,*48
$GPGSA,A,1,,,,,,,,,,,,,*1E
$GPRMC,002315.317,V,,,,,0.00,0.00,060180,,,N*42
$GPVTG,0.00,T,,M,0.00,N,0.00,K,N*32
$GPGGA,002316.317,,,,,0,0,,,M,,M,,*4B
$GPGSA,A,1,,,,,,,,,,,,
```

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

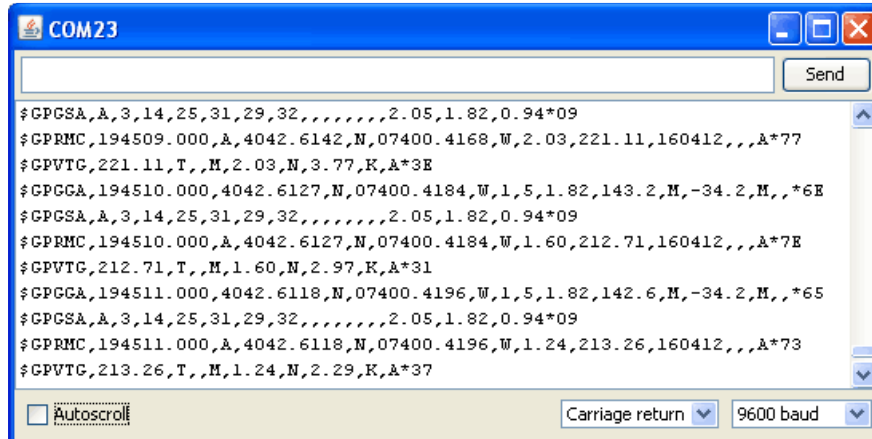
Recommended Minimum Coordinates or something like that) and the **\$GPGGA** sentences. These two provide the time, date, latitude, longitude, altitude, estimated land speed, and fix type. Fix type indicates whether the GPS has locked onto the satellite data and received enough data to determine the location (2D fix) or location+altitude (3D fix).

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

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 uFL plug) 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 \(https://adafru.it/aMI\)](https://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

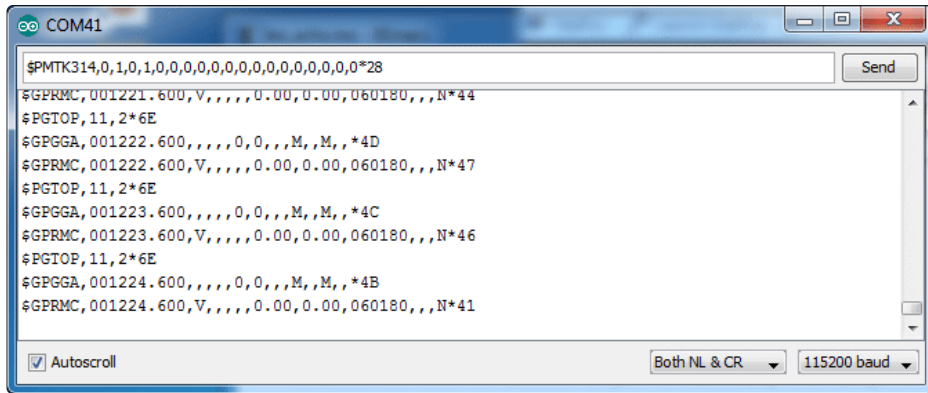
The one after that is **160412** which is the current date (16th of April, 2012).

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.

## Sending NMEA Commands via Direct Connect

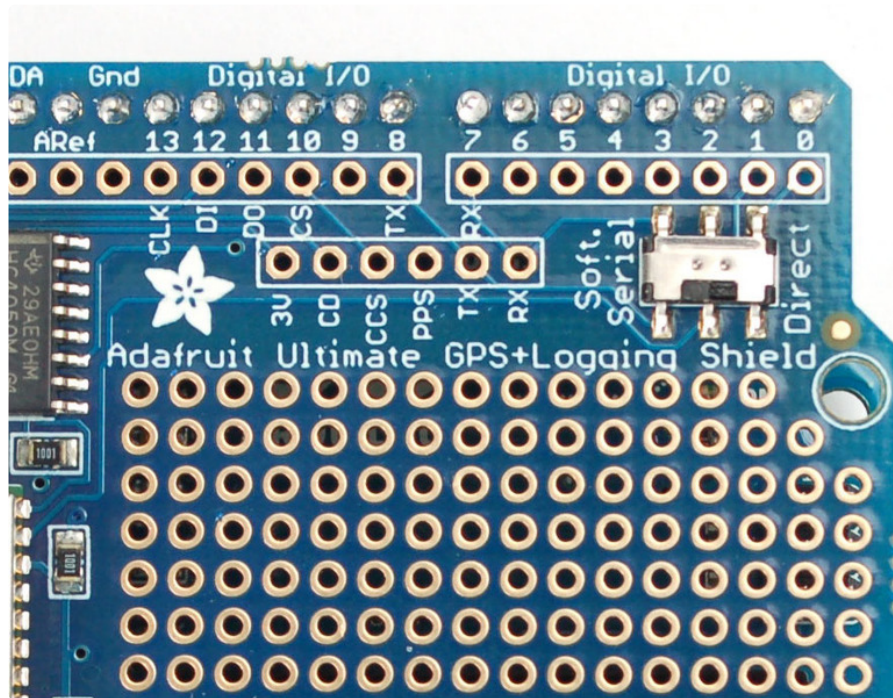
You can send \$PMTK and other commands from the GPS module datasheet, just type into the serial monitor box. Don't forget you'll need to set **Both NL & CR** (new line and carriage return) next to the baud rate selection box!



# Soft Serial Connect

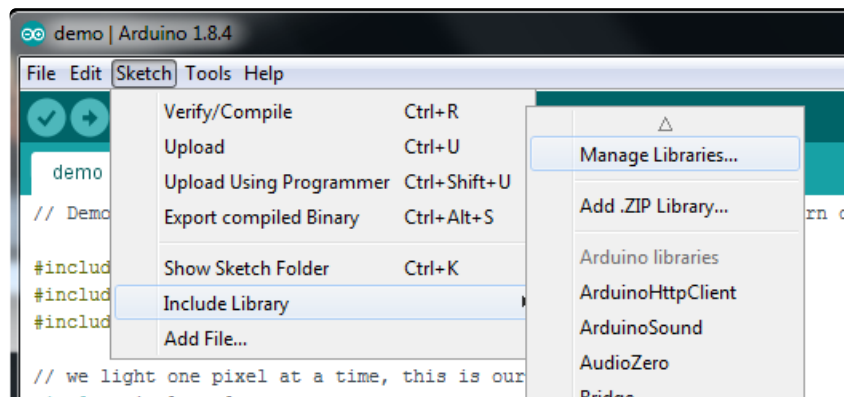
Once you've gotten the GPS module tested with direct wiring, we can go forward and set it up for Soft Serial connection. The soft serial connection works by setting up a secondary UART on two pins (digital 7 and 8) so that the main UART is free for debugging & uploading sketches

Soft Serial connection works on Uno/Duemilanove/Diecimila Arduinos as well as Leonardos. It does not work on Mega as the Mega does not have Soft Serial support on pins 7 & 8

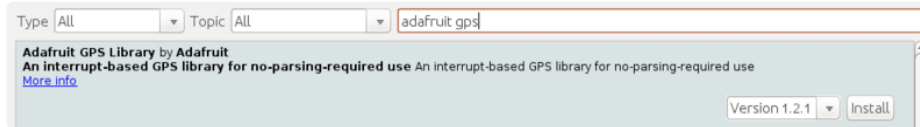


Next up, download the Adafruit GPS library. This library does a lot of the 'heavy lifting' required for receiving data from GPS modules, such as reading the streaming data in a background interrupt and automatically parsing it. This library can be downloaded from the Arduino library manager.

Open up the Arduino library manager:



Search for the **Adafruit GPS** library and install it



For Arduino UNO and other Atmega 328 boards: Due to limited memory, the shield\_sdlog example will not work with the latest version of the library. If you want to log GPS messages on the UNO, you will need to install version 1.3 or earlier of the GPS library.

We also have a great tutorial on Arduino library installation at:

<http://learn.adafruit.com/adafruit-all-about-arduino-libraries-install-use> (<https://adafru.it/aYM>)

Open up the **File→Examples→Adafruit\_GPS→leo\_echo** sketch and upload it to the Arduino. Then open up the serial monitor. This sketch simply reads data from the software serial port (pins 7&8) and outputs that to the hardware serial port connected to USB.

You can configure the 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.

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.



# Hardware Serial Connect

The current version of the GPS Logger Shield was designed quite some time ago now. Remember when the Arduino UNO was pretty much the only game in town? Well, as a result, it has some design features that are somewhat specific to the UNO and the boards of that era.

One of the more critical features is the switch that changes between "Soft. Serial" and "Direct". In order to better understand what it takes to use hardware serial, let's look at that switch in more detail.

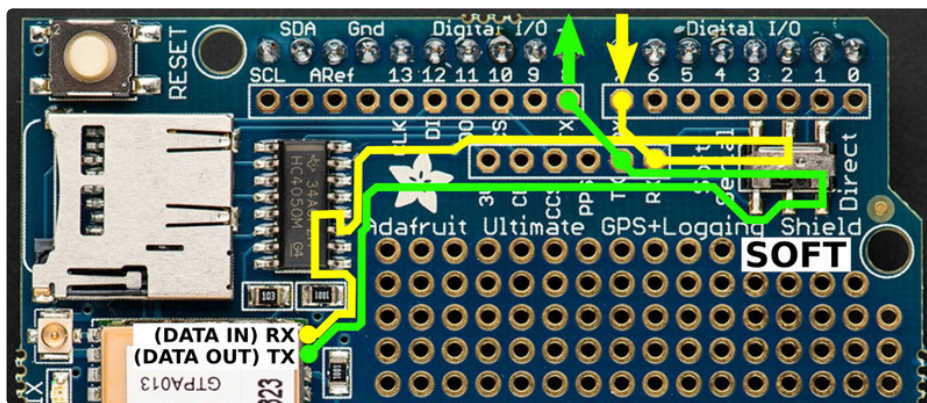
The Soft/Direct switch does nothing more than change the routing for the GPS's TX/RX pins.

## Soft Serial

When the switch is in this position, the GPS TX/RX pins are routed like this:

- GPS TX -> D8
- GPS RX -> D7

As well as the TX and RX pins on the auxiliary header row.



As long as your board can support software serial on those pins, then you're good to go.

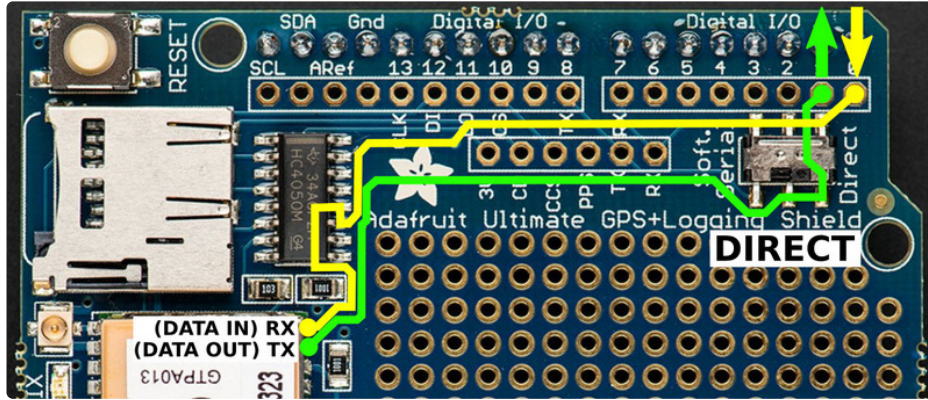
Not all boards support software serial on pins D8/D7.

## Direct Connect

When the switch is in this position, the GPS TX/RX pins are routed like this:

- GPS TX -> D1

- GPX RX -> D0



This mode was really meant to be used for general debugging.

While these two pins generally end up being where a board's hardware serial port shows up, the pins end up being backwards. The original idea with "Direct" was to allow **direct** connection between your host PC and the GPS unit. That meant routing the GPS's TX/RX to the USB-to-serial bridge's RX/TX, not the MCU's (ATmega 328, etc.) RX/TX.

"Direct" mode does not mean "Hardware Serial" mode.

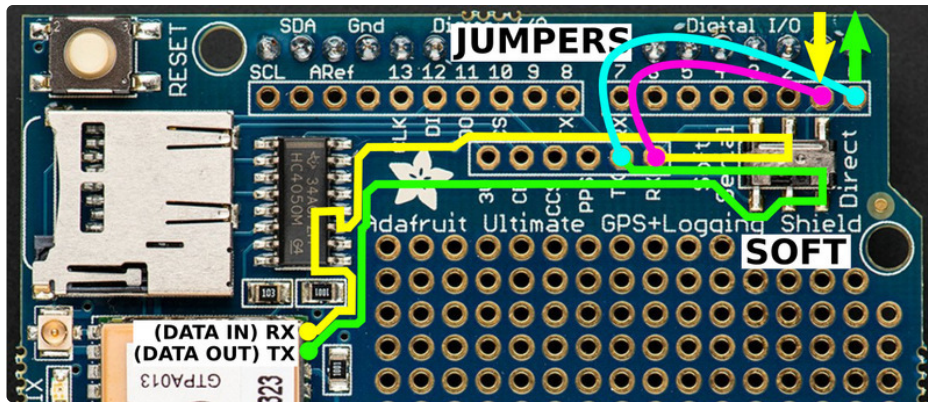
## Hardware Serial

OK, so how can you connect the GPS shield to a board's hardware serial pins? Well, it takes a bit of bodging, but can be done as follows. This is the same trick as described previously for the Leonardo, but here is provided in a more general way.

The trick is to set the switch to the "Soft. Serial" position. Then, add jumper wires from the auxiliary header row's TX/RX to pins 1/0 as follows:

- GPS TX -> D0
- GPS RX -> D1

Note how that ends up being the opposite of the "Direct" routing from above.

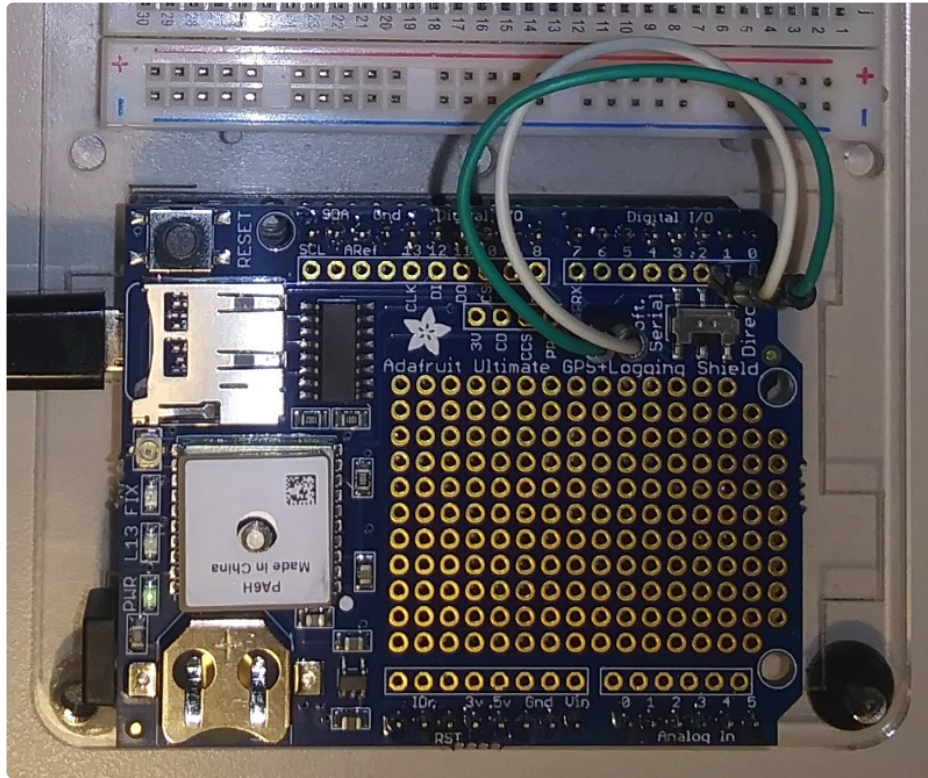


Yep - to use "hardware serial" the switch will be in the "soft serial" position.

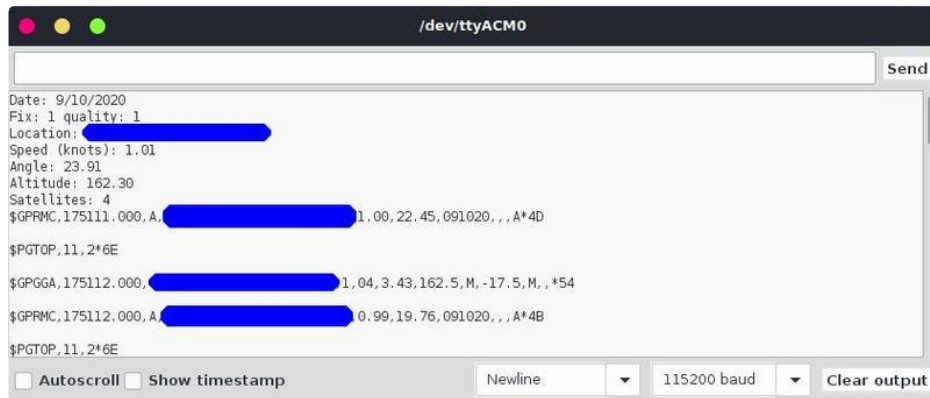
### Metro M0/M4 Example

The Metro M0 and M4 are good examples of boards where this trick can be used. Both the M0 and M4 MCU's have native USB support. That means there are dedicated pins that the host PC's USB can be directly connected to. No need for a USB-to-serial bridge solution, like the UNO uses. As a result, the hardware serial port provided on D0/D1 is completely separate.

Below we show the GPS Shield on top of a **Metro M4 Express**. The switch is set to "Soft.Serial" and the green/white wires are connected as described above.



And then we can run the [Hardware Serial Parsing \(https://adafru.it/TGa\)](https://adafru.it/TGa) example from the [GPS Library \(https://adafru.it/nCR\)](https://adafru.it/nCR). Here is what the serial monitor output looks like:



# Parsing 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.

## If you're using an Adafruit Metro 328P or Arduino Uno (or older)

Make sure the switch is set to SoftSerial

Open up the **File**→**Examples**→**Adafruit\_GPS**→**parsing** sketch and change the line

```
SoftwareSerial mySerial(3, 2);
```

to

```
SoftwareSerial mySerial(8, 7);
```

and upload it to the Arduino. Then open up the serial monitor.

## If you're using an Arduino Leonardo..

Make sure the switch is set to SoftSerial

Change **SoftwareSerial mySerial(3, 2);** to **SoftwareSerial mySerial(8, 7);** as above, and also change

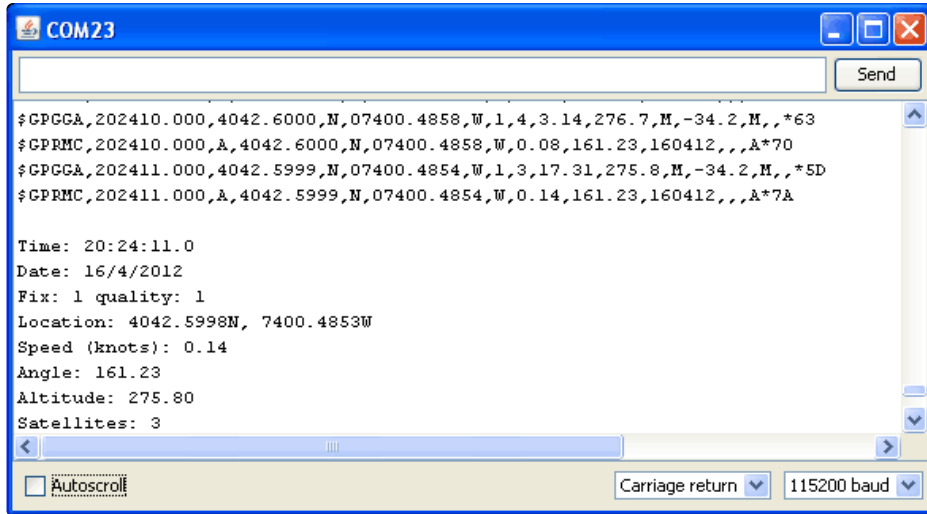
```
useInterrupt(true);
```

to

```
useInterrupt(false);
```

Because due to strange internal-details stuff, we cant use Software Serial, interrupts and also echo the output via USB at the same time on Leonardo.

You'll want to wait till you get a fix so stick the GPS out the window or use an antenna



In this sketch, we can either use interrupts and call **GPS.read()** within a once-a-millisecond timer (this is the same timer that runs the **millis()** command) or we check **GPS.read()** in the main loop constantly. 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 meters) and **GPS.satellites** (number of satellites)

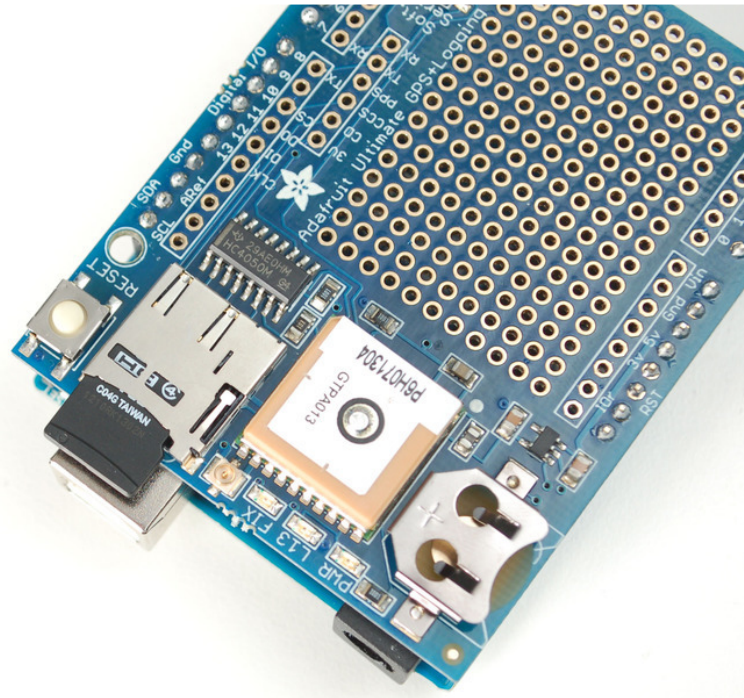
This should make it much easier to have location-based projects. We suggest keeping the update rate at 1Hz and request that the GPS only output RMC and GGA as the parser does not keep track of other data anyways.

# SD Logging

Finally we get to the fun stuff! Its time to log data from the GPS onto an SD card. First up we'll do a basic test to make sure the SD card system is working.

You'll need a microSD card to log data onto. Any capacity will do. Slide the microSD card into the silver socket. You can push it in and you'll feel it latch and click

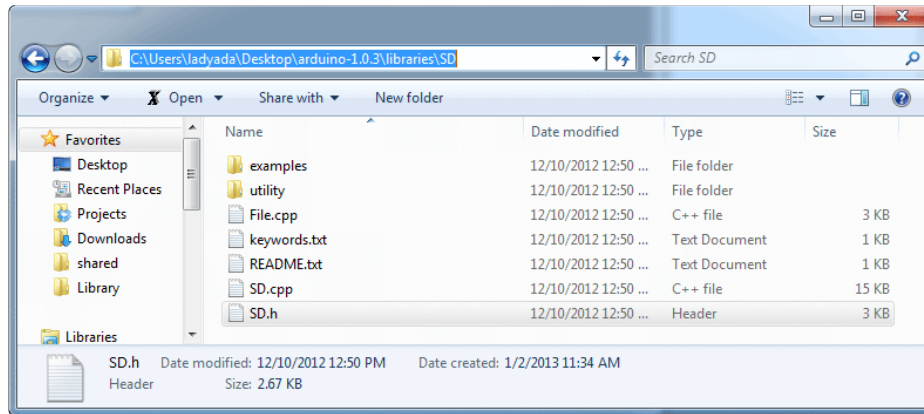
To remove, push on the edge and it will pop out (be careful it doesn't fly across the room!)



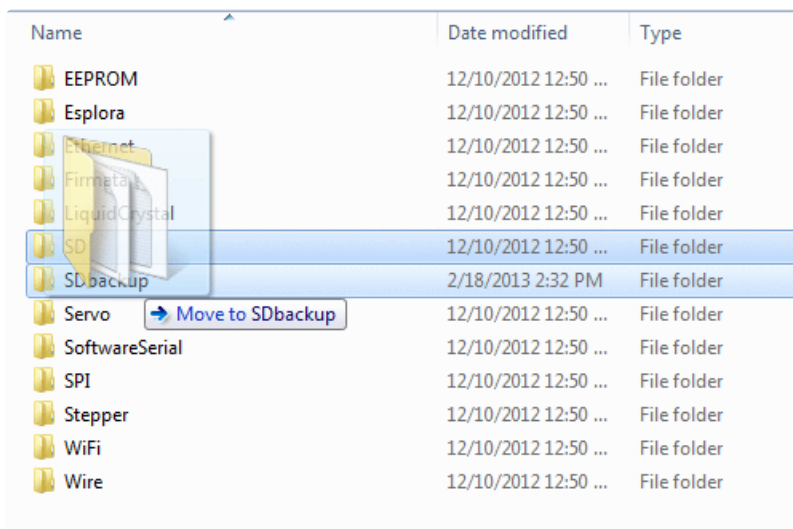
## SD Card Library Update

If you are using an Leonardo or Mega, you will have to update the SD card library to add 'SD card on any pin' support. If you have an Uno/Duemilanove/Diecimila, this is not required.

First, find the "core libraries" folder - if you are using Windows or Linux, it will be in the folder that contains the **Arduino** executable, look for a **libraries** folder. Inside you will see an **SD** folder (inside that will be **SD.cpp** **SD.h** etc)



In the **libraries** folder, make a new folder called **SDbackup**. Then drag the **SD** folder into **SDbackup**, this will 'hide' the old **SD** library without deleting it



Now we'll grab the new SD library, visit <https://github.com/adafruit/SD> (<https://adafru.it/aP6>) and click the **ZIP** download button, or click the button below

<https://adafru.it/cxl>

<https://adafru.it/cxl>

Uncompress and rename the uncompressed folder **SD**. Check that the **SD** folder contains **SD.cpp** and **SD.h**

Place the **SD** library folder your sketchbook libraries folder. You may need to create the libraries subfolder if its your first library. For more details on how to install libraries, [check out our ultra-detailed tutorial at \(https://adafru.it/aYM\)](https://adafru.it/aYM)<http://learn.adafruit.com/adafruit-all-about-arduino-libraries-install-use> (<https://adafru.it/aYM>)

Once installed, restart the IDE. Then open up the **Adafruit\_GPS->shield\_sdlog** sketch and upload it to your Arduino.



If you're using a Leonardo, uncomment the line

```
// while (!Serial);
```

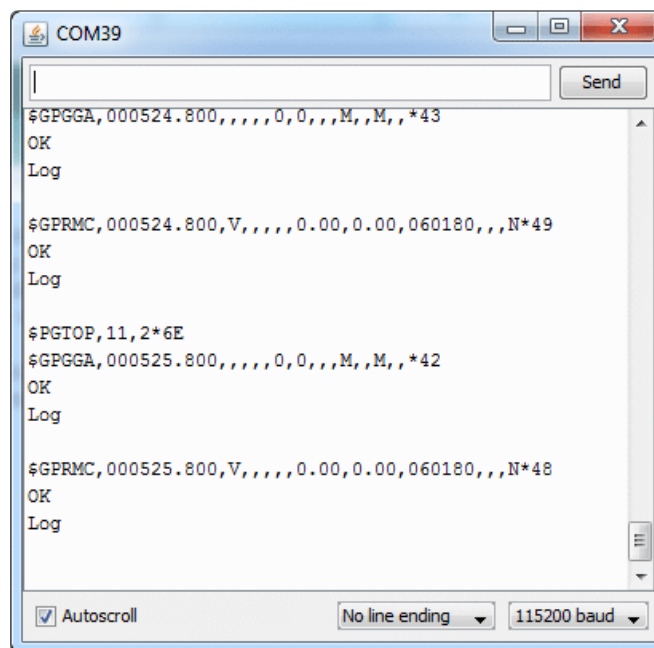
in the top of the **Setup()** function - this will help make debugging easier.

If you are using an Uno/Duemilanove/Diecimila or compatible, Find these two lines

```
if (!SD.begin(chipSelect, 11, 12, 13)) {  
//if (!SD.begin(chipSelect)) { // if you're using an UNO, you can use this line  
instead
```

And comment out the first one and uncomment the second one.

Make sure you have the switch set to **Soft Serial** and open up the serial monitor and check that you're connected at 115200 baud (or you'll get gibberish)



If the SD card was initialized OK, it will read a sentence from the GPS, check the checksum and if its a proper sentence, and then log it.

You can customize this basic example by changing the **sendCommand** in the top of **setup()**, for example you can turn on different sentences (although RMC and GGA are the most desirable, we've found). Or change the update rate to 5Hz instead of 1Hz

You can also set it so only 'fixed' data is logged - that is, there has to be valid location data. Doing so reduces power usage and saves space but its harder to debug because you don't get the timestamps that

are in the RMC sentences.

# Built In Logging

The built in logging capability isn't easy to use with the Shield - we wanted to make it better for SD logging so while we do have some details here, its not well supported and not recommended

For more details, you can read the [LOCUS \(built-in-datalogging system\) user guide \(https://adafru.it/uoc\)](https://adafru.it/uoc)

## Built In Logging

One of the nice things about the GPS module on the shield 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 some projects that want to track location, this can be a great low power way to log data - no SD card required! It can store up to ~16 hours of data.

The GPS module does require the Arduino 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

First, we should try getting the logger to run. Make sure the switch is set to SoftSerial

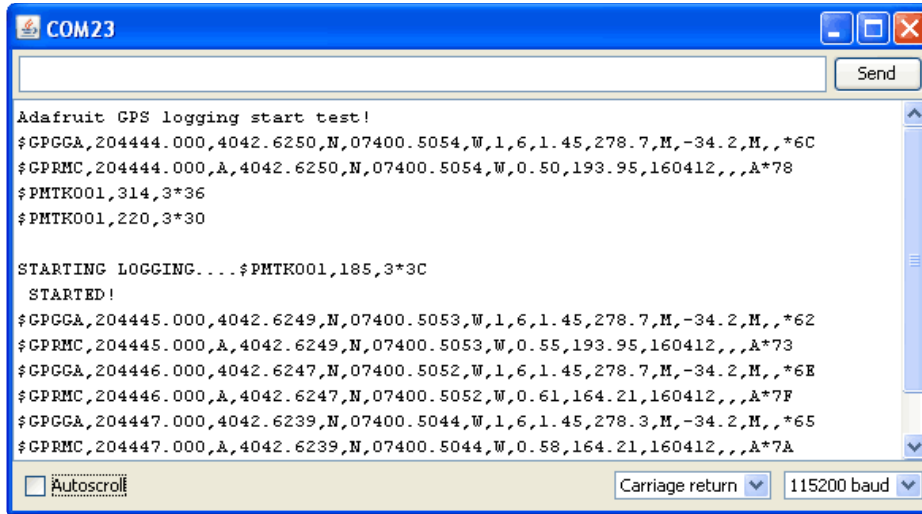
Open up the **File**→**Examples**→**Adafruit\_GPS**→**locus\_start** sketch and change the line

```
SoftwareSerial mySerial(3, 2);
```

to

```
SoftwareSerial mySerial(8, 7);
```

and upload it to the Arduino. Then open up the serial monitor. If you have an Uno or compatible you will see the echo'd data. Leonardos will not see the echo data but you should still see a message that says "STARTING LOGGIN...STARTED!"



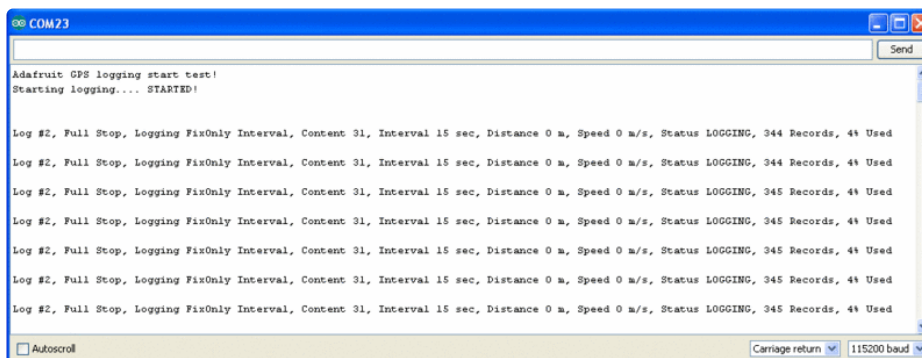
## Logging Status

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\_status** and change the line

**SoftwareSerial mySerial(3, 2);**

to

**SoftwareSerial mySerial(8, 7);**



## 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. Upload **File**→**Examples**→**Adafruit\_GPS**→**locus\_dump** to the Arduino and open up the serial monitor. Change the software serial line to use pins 8 and 7

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 (<https://adafru.it/aYN>) 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](#) (<https://adafru.it/cFg>).

## 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.

You can only do this with an Uno/Duemilanove/Diecimila/Mega!

You'll also need to set up direct wiring. Go back to the [connect with Direct Wiring example](#) (<https://adafru.it/aYO>), and get that working. You'll FTDI adapter or other TTL converter and download the [GPS Tool](#) (<https://adafru.it/aOQ>) - 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 \(https://adafru.it/cFg\)](https://adafru.it/cFg)



# Antenna, Battery and More!

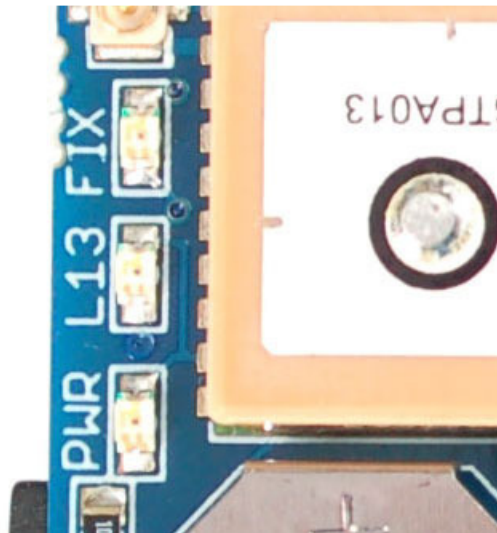
## LEDs

We love LEDs! There are three LEDs on board to help you with debugging and status updates:

Green **PWR** LED tells you that there is a good 3V power supply. If this isn't on, there's a serious problem with the power supply, perhaps the battery died

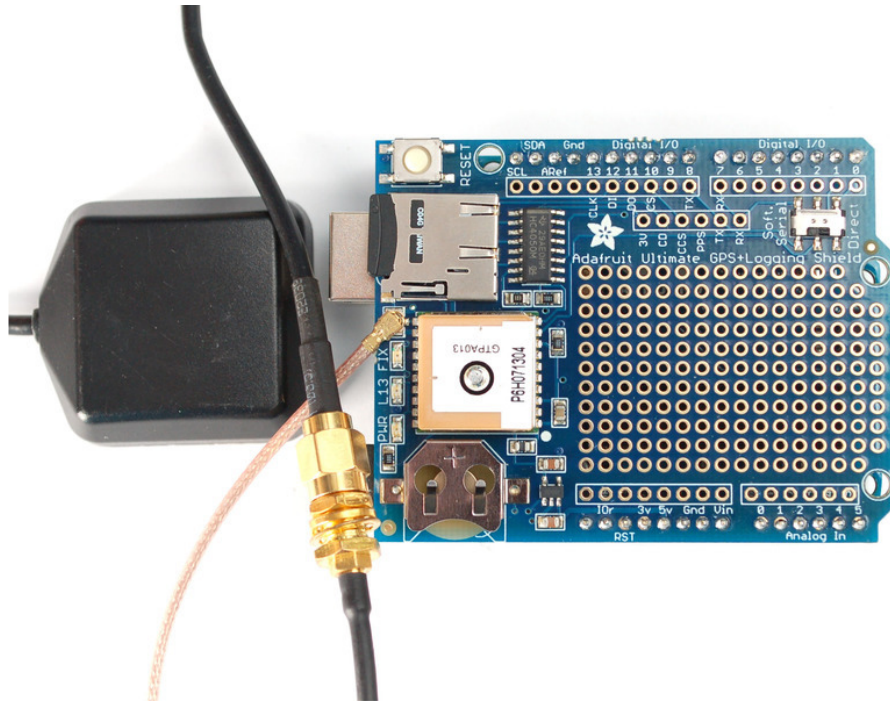
Yellow **L13** (and SD card access) LED is connected to digital 13, this is handy for telling when the Arduino is bootloading and also will flicker whenever the SD card is accessed.

Red **FIX** LED is connected to the GPS's fix output. When this is turning on/off once a second it **does not** have a fix. When it blinks once every 15 seconds, the GPS has a fix.



## Antennas

All Ultimate GPS shields 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) (<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. \(http://adafru.it/960\)](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 \(http://adafru.it/851\)](http://adafru.it/851). Then connect the GPS antenna to the cable.

The Ultimate GPS shield 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.

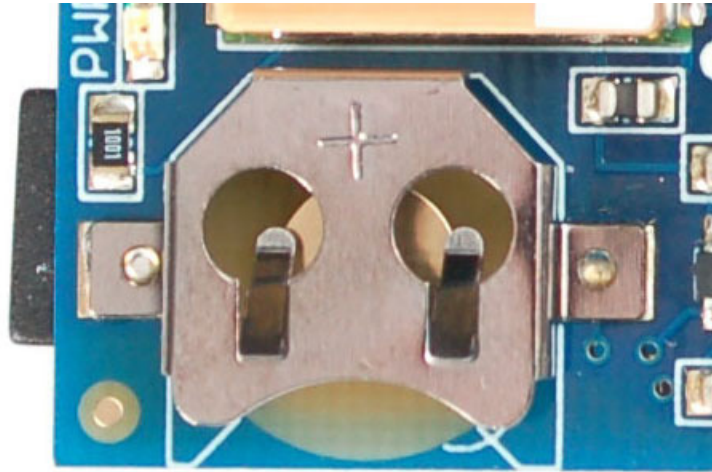


```
COM91
$GPGSA,A,1,,,,,,,,,,,,,*1E
$GPRMC,000101.799,V,,,,,0.00,0.00,060180,,,N*45
$GPVTG,0.00,T,,M,0.00,N,0.00,K,N*32
$PGTOP,11,3*6F
$GPGGA,000102.799,,,,,0,0,,,M,M,,*4C
$GPGSA,A,1,,,,,,,,,,,,,*1E
$GPRMC,000102.799,V,,,,,0.00,0.00,060180,,,N*46
$GPVTG,0.00,T,,M,0.00,N,0.00,K,N*32
$PGTOP,11,3*6F
$GPGGA,000103.799,,,,,0,0,,,M,M,,*4D
$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
```

## RTC Backup Battery

The Ultimate GPS module does not have a built in RTC battery, but a coin cell is provided, simply place the coin cell inside the holder, with the + facing up, thats it! The battery will last 7+ years

The real time clock will *automatically* set itself to the correct UTC time as soon as the GPS gets its first correct signal from a satellite. Without the battery, if the GPS loses power, it will forget the time and has to wait till it gets signal again to reset. But, if you have the battery in, it will keep time even after a power loss. We recommend it highly!



## Breakout and Proto-area

Since the GPS module is so small, we had tons of space left over, we turned that into a proto area. Feel free to put any circuitry you'd like into the 0.1" spaced holes. None of the holes are connected.

There's also a few breakouts near the SoftSerial/Direct switch. For example, the **PPS** pin which provides pulse-per-second output when the GPS has a fix.

- **3V** - this is the output from the onboard 3V regulator, it provides ~100mA if you need another 3V supply
- **CD** - this is the card-detect output from the microSD socket, it is shorted to ground when a card is **not** inserted. You'll want to have a pullup if you plan to use this pin to detect if a card is inserted
- **CCS** - this is another breakout for the Card Chip Select line. By default its connected to digital 10 but if you want to change it you can cut the trace from here to #10 and then solder in a new wire
- **PPS** - pulse-per-second output when the GPS has a fix
- **TX** and **RX** - extra 'copies' of the GPS RX/TX pins - the RX pin is 5V friendly.

# Downloads

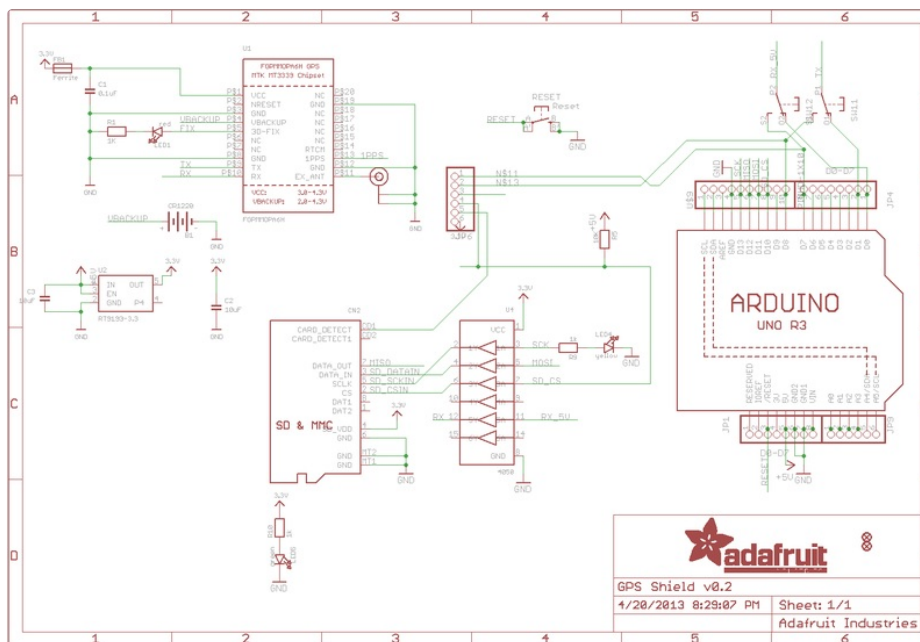
There's also more information at the GPS module tutorial page <http://learn.adafruit.com/adafruit-ultimate-gps> (<https://adafru.it/aTI>)

- [MTK3329/MTK3339 command set sheet](https://adafru.it/e7A) (<https://adafru.it/e7A>) for changing the fix data rate, baud rate, sentence outputs, etc!
- [Datasheet for the PA6H \(MTK3339\) GPS module itself](https://adafru.it/aPO) (<https://adafru.it/aPO>)
- [MT3339 GPS PC Tool \(windows only\)](https://adafru.it/uof) (<https://adafru.it/uof>) and the [PC Tool manual](https://adafru.it/uoA) (<https://adafru.it/uoA>)
- [Sample code and spec sheet for the LOCUS built-in logger](https://adafru.it/aTi) (<https://adafru.it/aTi>)
- [Mini GPS tool \(windows only\)](https://adafru.it/aMs) (<https://adafru.it/aMs>)
- [LOCUS \(built-in-datalogging system\) user guide](https://adafru.it/uoc) (<https://adafru.it/uoc>)
- [EagleCAD PCB files on GitHub](https://adafru.it/s9C) (<https://adafru.it/s9C>)
- [Fritzing object in the Adafruit Fritzing Library](https://adafru.it/aP3) (<https://adafru.it/aP3>)

More reading:

- [Trimble's GPS tutorial](https://adafru.it/emh) (<https://adafru.it/emh>)
- [Garmin's GPS tutorial](https://adafru.it/aMv) (<https://adafru.it/aMv>)

## Schematic



## 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 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 (<https://adafru.it/dR5>), you can [use this tool to upload it using an FTDI or USB-TTL cable \(or direct wiring with FTDI\)](#) (<https://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!

---

## 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 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!

---

## 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="")
```



