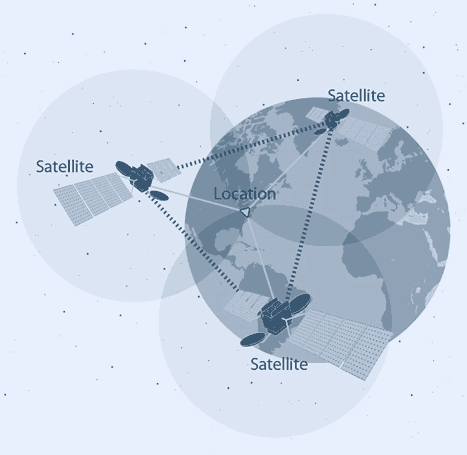
**NEO-6M GPS Module Interfacing with Arduino**

### How does GPS work?

GPS receivers actually work by figuring out how far they are from a number of satellites. They are pre-programmed to know where the GPS satellites are at any given time.

The satellites transmit information about their position and the current time in the form of radio signals towards the Earth. These signals identify the satellites and tell the receiver where they are located.



The receiver then calculates how far away each satellite is by figuring out how long it took for the signals to arrive. Once it has information on how far away at least three satellites are and where they are in space, it can pinpoint your location on Earth.

This process is known as Trilateration.

****

### NEO-6M GPS Chip

At the heart of the module is a NEO-6M GPS chip from u-blox. The chip measures less than the size of a postage stamp but packs a surprising amount of features into its little frame.

It can track up to 22 satellites on 50 channels and achieves the industry’s highest level of sensitivity i.e. -161 dB tracking, while consuming only 45mA supply current.

Unlike other GPS modules, it can do up to 5 location updates a second with 2.5m Horizontal position accuracy. The u-blox 6 positioning engine also boasts a **Time-To-First-Fix** (TTFF) of under 1 second.

One of the best features the chip provides is Power Save Mode(PSM). It allows a reduction in system power consumption by selectively switching parts of the receiver ON and OFF. This dramatically reduces power consumption of the module to just **11mA** making it suitable for power sensitive applications like GPS wristwatch.

The necessary data pins of NEO-6M GPS chip are broken out to a 0.1″ pitch headers. This includes pins required for communication with a microcontroller over UART. The module supports baud rate from 4800bps to 230400bps with default baud of 9600.

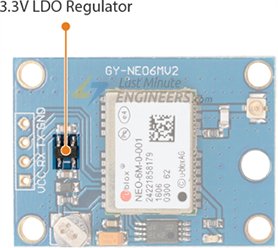
Here are complete specifications:

|  |  |
| --- | --- |
| Receiver Type | 50 channels, GPS L1(1575.42Mhz) |
| Horizontal Position Accuracy | 2.5m |
| Navigation Update Rate | 1HZ (5Hz maximum) |
| Capture Time | Cool start: 27sHot start: 1s |
| Navigation Sensitivity | -161dBm |
| Communication Protocol | NMEA, UBX Binary, RTCM |
| Serial Baud Rate | 4800-230400 (default 9600) |
| Operating Temperature | -40°C ~ 85°C |
| Operating Voltage | 2.7V ~ 3.6V |
| Operating Current | 45mA |
| TXD/RXD Impedance | 510Ω |

There is an LED on the NEO-6M GPS Module which indicates the status of Position Fix. It’ll blink at various rates depending on what state it’s in:

* **No Blinking** – It’s searching for satellites.
* **Blink every 1s** – Position Fix is found(The module can see enough satellites).

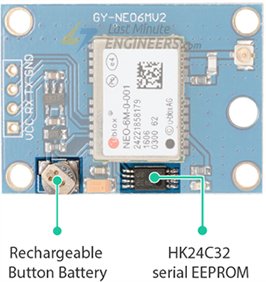
### 3.3V LDO Regulator



The operating voltage of the NEO-6M chip is from 2.7 to 3.6V. But the good news is that, the module comes with MIC5205 ultra-low dropout 3V3 regulator from [MICREL](https://www.mouser.com/ds/2/268/mic5205-778789.pdf).

The logic pins are also 5-volt tolerant, so we can easily connect it to an Arduino or any 5V logic microcontroller without using any logic level converter.

### Battery & EEPROM



The module is equipped with an **HK24C32** two wire serial EEPROM. It is 4KB in size and connected to the NEO-6M chip via I2C.

The module also contains a rechargeable button battery which acts as a super-capacitor.

An EEPROM together with battery helps retain the **battery backed RAM** (BBR). The BBR contains clock data, latest position data(GNSS orbit data) and module configuration. But it’s not meant for permanent data storage.

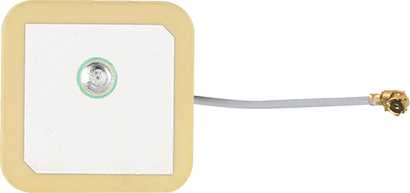
As the battery retains clock and last position, time to first fix (TTFF) significantly reduces to 1s. This allows much faster position locks.

Without the battery the GPS always cold-start so the initial GPS lock takes more time.

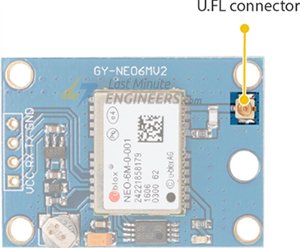
The battery is automatically charged when power is applied and maintains data for up to **two weeks** without power.

### Antenna

An antenna is required to use the module for any kind of communication. So, the module comes with a patch antenna having -161 dBm sensitivity.



You can snap-fit this antenna to small U.FL connector located on the module.

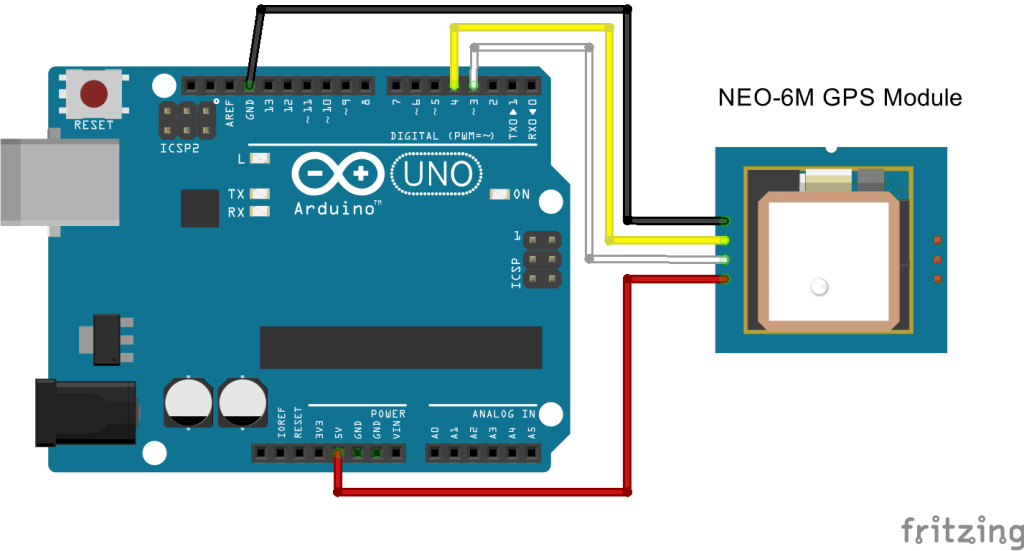


Patch antenna is great for most projects. But if you want to achieve more sensitivity or put your module inside a metal case, you can also snap on any 3V active GPS antenna via the U.FL connector.

The NEO-6M GPS module is shown in the figure below. It comes with an external antenna, and does’t come with header pins. So, you’ll need to get and solder some.Features of this module are:

* This module has an external antenna and built-in EEPROM.
* Interface: RS232 TTL
* Power supply: 3V to 5V
* Default baudrate: 9600 bps
* Works with standard NMEA sentences
* Module is compatible with other microcontroller boards.

### Pin Wiring



The NEO-6M GPS module has four pins: VCC, RX, TX, and GND. The module communicates with the Arduino via serial communication using the TX and RX pins, so the wiring couldn’t be simpler:

|  |  |
| --- | --- |
| **NEO-6M GPS Module** | **Wiring to Arduino UNO** |
| VCC | 3.3V |
| RX | TX pin defined in the software serial |
| TX | RX pin defined in the software serial |
| GND | GND |

## Getting GPS Raw Data

To get raw GPS data you just need to start a serial communication with the GPS module using Software Serial. Continue reading to see how to do that.

### Parts Required

For testing this example you’ll need the following parts:

* Arduino
* NEO-6M GPS Module
* Jumper Wires

Connections with Arduino

* The module GND pin is connected to Arduino GND pin
* The module RX pin is connected to Arduino pin 3
* The module TX pin is connected to Arduino pin 4
* The module VCC pin is connected to Arduino 5V pin

### Code

Copy the following code to your Arduino IDE and upload it to your Arduino board.

/\*

\*

\*/

#include <SoftwareSerial.h>

// The serial connection to the GPS module

SoftwareSerial ss(4, 3);

void setup(){

Serial.begin(9600);

ss.begin(9600);

}

void loop(){

while (ss.available() > 0){

// get the byte data from the GPS

byte gpsData = ss.read();

Serial.write(gpsData);

}

### Understanding NMEA Sentences

NMEA sentences start with the $ character, and each data field is separated by a comma.

**$GPGGA**,110617.00,41XX.XXXXX,N,00831.54761,W,1,05,2.68,129.0,M,50.1,M,,\*42

**$GPGSA**,A,3,06,09,30,07,23,,,,,,,,4.43,2.68,3.53\*02

**$GPGSV**,3,1,11,02,48,298,24,03,05,101,24,05,17,292,20,06,71,227,30\*7C

**$GPGSV**,3,2,11,07,47,138,33,09,64,044,28,17,01,199,,19,13,214,\*7C

**$GPGSV**,3,3,11,23,29,054,29,29,01,335,,30,29,167,33\*4E

**$GPGLL**,41XX.XXXXX,N,00831.54761,W,110617.00,A,A\*70

**$GPRMC**,110618.00,A,41XX.XXXXX,N,00831.54753,W,0.078,,030118,,,A\*6A

**$GPVTG**,,T,,M,0.043,N,0.080,K,A\*2C

There are different types of NMEA sentences. The type of message is indicated by the characters before the first comma.

The GP after the $ indicates it is a GPS position.  The $GPGGA is the basic GPS NMEA message, that provides 3D location and accuracy data. In the following sentence:

Parsing NMEA Sentences

There are many sentences in the NMEA standard, the most common ones are:

* $GPRMC (Global Positioning Recommended Minimum Coordinates) provides the time, date, latitude, longitude, altitude and estimated velocity.
* $GPGGA sentence provides essential fix data which provide 3D location and accuracy data.

Let’s take an example of $GPRMC NMEA sentence from a GPS receiver.

$GPRMC, 123519, A, 4807.038, N, 01131.000, E,022.4, 084.4, 230394, 003.1, W\*6A

|  |  |
| --- | --- |
| $ | Every NMEA sentence starts with $ character. |
| GPRMC | Global Positioning Recommended Minimum Coordinates |
| 123519 | Current time in UTC – 12:35:19 |
| A | Status A=active or V=Void. |
| 4807.038,N | Latitude 48 deg 07.038′ N |
| 01131.000,E | Longitude 11 deg 31.000′ E |
| 022.4 | Speed over the ground in knots |
| 084.4 | Track angle in degrees True |
| 220318 | Current Date – 22rd of March 2018 |
| 003.1,W | Magnetic Variation |
| \*6A | The checksum data, always begins with \* |

Let’s take an example of $GPGGA NMEA sentence.

$GPGGA, 123519, 4807.038, N, 01131.000, E, 1, 08, 0.9, 545.4, M, 46.9, M, , \*47

|  |  |
| --- | --- |
| $ | Starting of NMEA sentence. |
| GPGGA | Global Positioning System Fix Data |
| 123519 | Current time in UTC – 12:35:19 |
| 4807.038,N | Latitude 48 deg 07.038′ N |
| 01131.000,E | Longitude 11 deg 31.000′ E |
| 1 | GPS fix |
| 08 | Number of satellites being tracked |
| 0.9 | Horizontal dilution of position |
| 545.4,M | Altitude in Meters (above mean sea level) |
| 46.9,M | Height of geoid (mean sea level) |
| (empty field) | Time in seconds since last DGPS update |
| (empty field) | DGPS station ID number |
| \*47 | The checksum data, always begins with \* |

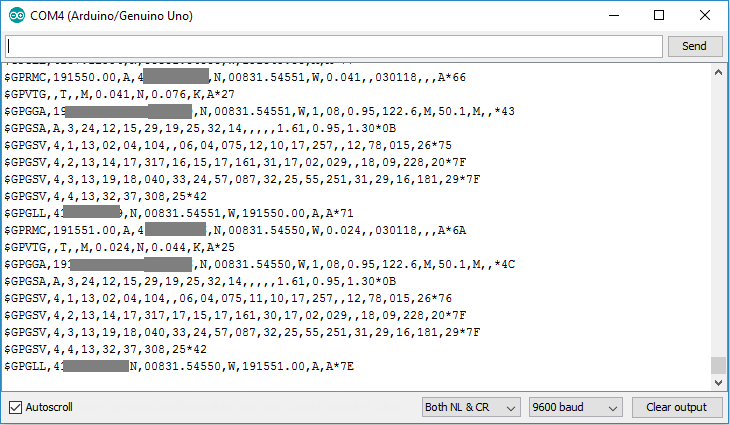
For more information about NMEA sentences and what data they contain,

**$GPGGA**,110617.00,41XX.XXXXX,N,00831.54761,W,1,05,2.68,129.0,M,50.1,M,,\*42

* **110617** – represents the time at which the fix location was taken, 11:06:17 UTC
* **41XX.XXXXX,N** – latitude 41 deg XX.XXXXX’ N
* **00831.54761,W** – Longitude 008 deg 31.54761′ W
* **1** – fix quality (0 = invalid; 1= GPS fix; 2 = DGPS fix; 3 = PPS fix; 4 = Real Time Kinematic; 5 = Float RTK; 6 = estimated (dead reckoning); 7 = Manual input mode; 8 = Simulation mode)
* **05** – number of satellites being tracked
* **2.68** – Horizontal dilution of position
* **129.0, M** – Altitude, in meters above the sea level
* **50.1, M** – Height of geoid (mean sea level) above WGS84 ellipsoid
* empty field – time in seconds since last DGPS update
* empty field – DGPS station ID number
* **\*42** – the checksum data, always begins with \*

The other NMEA sentences provide additional information:

* **$GPGSA** – GPS DOP and active satellites
* **$GPGSV** – Detailed GPS satellite information
* **$GPGLL** – Geographic Latitude and Longitude
* **$GPRMC** – Essential GPS pvt (position, velocity, time) data
* **$GPVTG** – Velocity made good



Serial Monitor Output

**Using TinyGPS+ Library**

You can work with the raw data from the GPS, or you can convert those NMEA messages into a readable and useful format, by saving the characters sequences into variables. To do that, we’re going to use the [TinyGPS++ library](https://github.com/mikalhart/TinyGPSPlus).

This library makes it simple to get information on location in a format that is useful and easy to understand.

### Installing the TinyGPS++ Library

Follow the next steps to install the TinyGPS++ library in your Arduino IDE:

1. Download the TinyGPSPlus library. You should have a .zip folder in your Downloads folder
2. Unzip the .zip folder and you should get TinyGPSPlus-master folder
3. Rename your folder from  to TinyGPSPlus
4. Move the TinyGPSPlus folder to your Arduino IDE installation libraries folder
5. Finally, re-open your Arduino IDE

Copy the following code to your Arduino IDE and upload it to your Arduino board.

#include <TinyGPS++.h>

#include <SoftwareSerial.h>

static const int RXPin = 4, TXPin = 3;

static const uint32\_t GPSBaud = 9600;

// The TinyGPS++ object

TinyGPSPlus gps;

// The serial connection to the GPS device

SoftwareSerial ss(RXPin, TXPin);

void setup(){

Serial.begin(9600);

ss.begin(GPSBaud);

}

void loop(){

// This sketch displays information every time a new sentence is correctly encoded.

while (ss.available() > 0){

gps.encode(ss.read());

if (gps.location.isUpdated()){

Serial.print("Latitude= ");

Serial.print(gps.location.lat(), 6);

Serial.print(" Longitude= ");

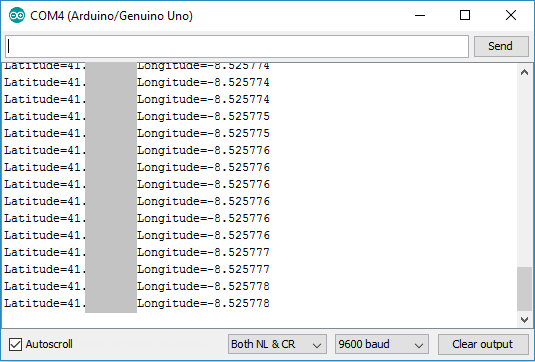
Serial.println(gps.location.lng(), 6);

}

}

}

Getting the latitude and longitude is has simple has using gps.location.lat(), and gps.location.lng(), respectively.Upload the code to your Arduino, and you should see the location displayed on the serial monitor. After uploading the code, wait a few minutes while the module adjusts the position to get a more accurate data.



Serial Monitor Output