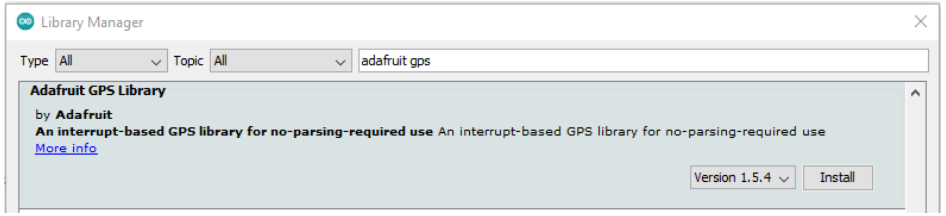
Using the Adafruit GPS Library

LaACES Student Ballooning Course

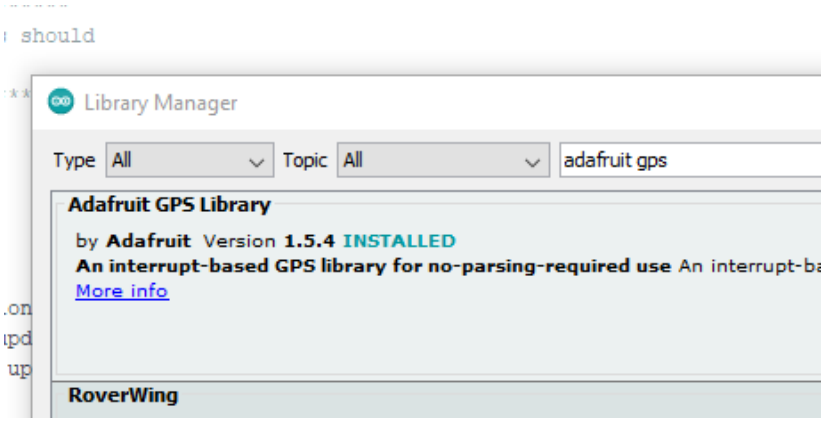
Installing the Library

- Easiest way to install is using the Arduino Library Manager
  - Select Tools->Manage Libraries
- We can also manually install by downloading the .zip file and unpacking the library into the sketchbook libraries folder
  - https://github.com/adafruit/Adafruit_GPS
  - Remember we set the Sketchbook in the preferences
- As of October 2021 most recent version is 1.5.4



After installation

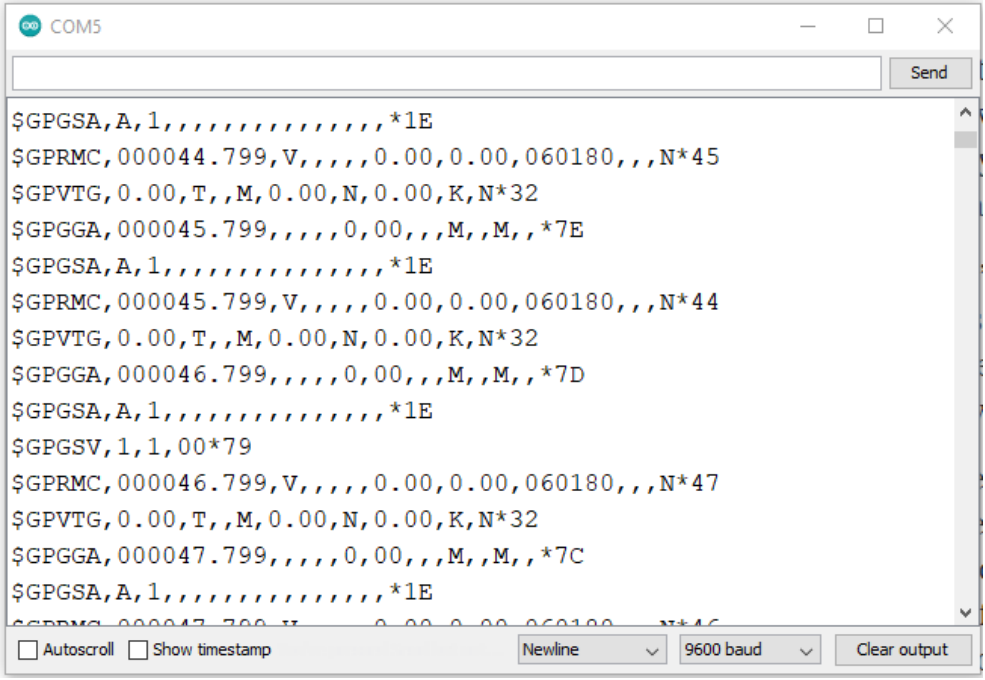
- After installation should show up as installed in the library manager and library should be listed under the File-> Examples and Sketch -> Include Library menu options
- We need to have the line #include <Adafruit_GPS.h> at the top of our sketch to use the library
  - Adafruit_GPS.h should be orange if the library was installed correctly




```
/* Include statements for requi
#include <Adafruit_GPS.h>
/* Adafruit Ultimate GPS Logger
#define GPSSerial Serial1
Adafruit_GPS GPS(&GPSSerial);
```

Communicating with the GPS

- We send and receive data from the GPS using plain text strings
- These strings are are sent and received as ASCII characters over UART at 9600 baud via the TX and RX pins of the shield
  - Baud rate can be changed, but default is 9600
- Remember was can switch which Arduino pins using the Direct/Soft-Serial Switch



Data Streams

- Data received from the GPS shield will be NMEA sentences.
  - Figure 10 shows what raw GPS data will look like in the Serial Monitor
- $GPRMC,212628.068,V,,,,,0.00,0.00,150819,,,N*4A
  - This is an example of a NMEA sentence pulled from Figure 10
  - From this data stream, we can see it was taken on August 15, 2019 at 21:26:28 UTC. A fix was not achieved.

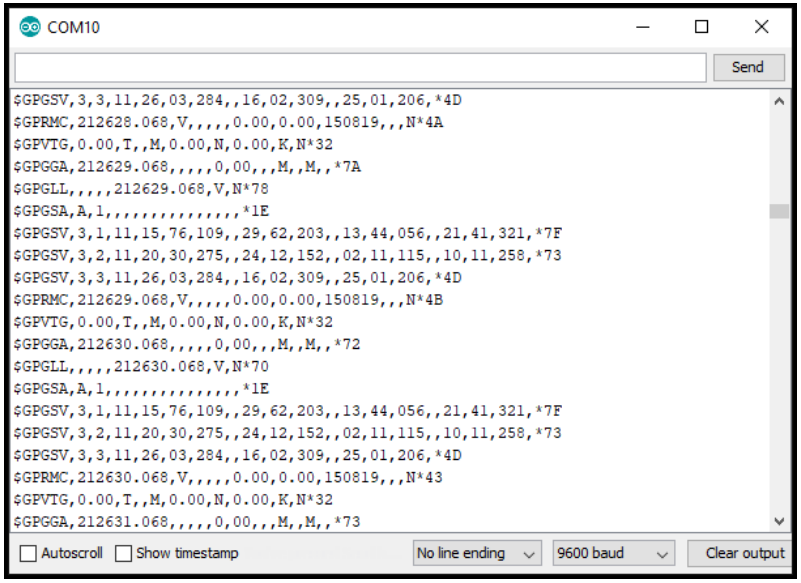


Figure 10: Shown is an example of the raw output of the Adafruit Ultimate GPS Logger Shield on the serial monitor when the Shield is switched to Direct Connect. The GPS did not have a fix, so most data fields are blank

Input/Output Strings

Output

- NMEA Sentences
  - All selected sentences
    - Default GGA,GSA, RMC, VTG sentences on
  - Automatically at set rate
    - Default every 1 second
- Will also send response string to any commands send

Input

- Commands take the form of preformatted text strings similar to NMEA
- Commands are called PMTK
  - MediaTek Protocol named after the GPS manufacturer
- Full Command list reference
  - https://laspace.lsu.edu/laaces/wp-content/uploads/2021/10/PMTK_Packet_User_Manual.pdf

GPS Object

- We use the GPS in programming by declaring a variable of the Adafruit_GPS type and we need to tell it which serial port the GPS is connected to
- Here GPS is the name of our variable
- The line above tells the compiler that we want Serial1 every place that put GPSSerial
- The & just tells that we want to send the location in memory of the Serial object instead of the actual Serial object
  - The way the GPS library is written we send the Serial this way

```
/* Adafruit Ultimate GPS Logge
#define GPSSerial Serial1
Adafruit_GPS GPS(&GPSSerial);
```

GPS Functions

- Once we have created the GPS object we call functions of it to do operations
- GPS.sendCommand() – sends the string inside the parenthesis to the GPS
- GPS.read() – reads a character off of the GPS serial, returns it and copies it to the current NMEA sentence being built
- GPS.lastNMEA() – returns the last complete NMEA received
- GPS.newNMEAreceived() – returns true or false if a new (since last time lastNMEA() was used) complete NMEA is available
- GPS.parse() – breaks apart the NMEA we give it inside the parenthesis and updates all of the individual GPS variables

GPS Data Flow

1. GPS outputs a characters to the Arduino Serial, all sentences are output with no pause (only time required to send by the baudrate)
2. Since we are connected to a Hardware Serial, Arduino Automatically copies the characters into the buffer for that serial port
   - This has space for 64 characters. If more are sent before they are read old characters will be overwritten
3. We call GPS.read(), this returns the next character out of the buffer and add it to the end “current NMEA” being built, if the character read is a newline (indicating the end of a sentence) 3 things happen:
   1. Copies “current NMEA into lastNMEA (overwriting the old one)
   2. Sets newNMEAreceived to be true
   3. Starts a new “current NMEA”

Reading the GPS

- GPS send data to a Serial buffer. This buffer needs to be read manually. There are two ways – polling and interrupts. Interrupts are the proper way.
- Polling
  - Whenever we get to certain point in the program we check to see if there are characters to read
    - If program gets busy can miss characters
- Interrupt
  - When a specific signal is received, the Arduino pauses what it is current doing and goes and reads the serial buffer. Otherwise, it keeps on doing other things.
    - Signal could be a timer, a signal on a pin, etc.

Interrupt Example

- Figure 9 is the code used to interrupt and read the GPS serial buffer
- Timer0 is an internal timer used for millis()
- This interrupt checks the serial buffer every millisecond regardless of what else the Arduino is doing
  - This ensure we don't miss a character

```
79  SIGNAL(TIMER0_COMPA_vect) {
80  /******* Interrupt Service Routine ************************************************
81   *  This is the interrupt service routine. It reads a character from the GPS.
82   *********************************************************************************/
83    char c = GPS.read();      // Read a character from the GPS
84  }
85
86
87  void useInterrupt(boolean v) {
88  /******* useInterrupt *************************************************************
89   *  This function is enables or disables the SIGNAL(TIMER0_COMPA_vect) interrupt.
90   *********************************************************************************/
91    // millis() uses Timer0, so we'll interrupt in the middle and call the function above
92    if (v) {
93      OCR0A = 0xAF;
94      TIMSK0 |= _BV(OCIE0A);
95      usingInterrupt = true;
96    }
97
98    // Do not call the interrupt function COMPA anymore
99    else {
100     TIMSK0 &= ~_BV(OCIE0A);
101     usingInterrupt = false;
102   }
103 }
```

Figure 9: The top function simply reads a character from the GPS serial buffer. The second function uses Timer0 (an internal timer on the Mega), which is used to increment the millis() call. This function says to interrupt between the millis() increments (so as not to interfere with it). This will result in the Mega reading a character from the GPS serial buffer once a millisecond.

PMTK Commands

Command Structure: String of characters similar to NMEA sentences

$PMTK###,<Command Data>*CS<CR><LF>

1. $ - Indicates Start of Command String
2. 'PMTK' – Character string indicating MediaTek Packet
3. '###' – 3 Digit character string of numbers indicating what the command is
4. <Command Data> - the actual command usually comma separated series of numbers
5. *CS – Character representation in hexidecimal of the checksum
6. <CR><LF> - Carriage return and line feed

Sending Commands

- Using GPS.sendCommand() we can send a string of characters that the GPS will treat as command
- We could send the entire string manually:
  - GPS.sendCommand("$PMTK314,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0*29"); This will send turn off all NMEA sentences except GLL
- The Library defines constants for most of the useful command strings so we do not have to figure out the entire string:
  - GPS.sendCommand(PMTK_SET_NMEA_OUTPUT_GLLONLY); This will accomplish the same thing as above
- After sending a command the GPS will respond with an acknowledge, this will be between the automatic NMEA sentences

Setting Balloon Mode

- Some GPS shields have had their altitude lockup at ~10km.
- This might be due to some units being set a navigation mode that limits the altitude to 10km
- To set the unit to “Balloon Mode” be sure to include the following line in your setup:
  - GPS.sendCommand(“$PMTK886,3*2B”);
  - The Adafruit documentation does not discuss this but we think this may be the cause
  - We found a more recent version of the PMTK command list that includes this
  - Remember that the GPS will return to default on power loss

Parsing

- By calling GPS.Parse() the GPS library will break down the NMEA sentence into the individual data pieces and update internal variables that we can then read the values of
  - We do need to pass the NMEA to the Parse() function for example like -> GPS.Parse(GPS.LastNMEA());
- For example, after we a sentence that contained an altitude reading 300.2 meters we can call GPS.altitude and we will get 300.2
- It is important to remember that these values only get updated when we call Parse on a NMEA sentence that has those variables.
  - For example, if we have only called Parse on GGA sentences (which do not include the year) and we call GPS.year we will get the default year value
  - We also need a fix to get an accurate value

GPS Command - Type	Returns
GPS.latitude – float	Latitude (Degrees and Minutes DDMM.MMMM)
GPS.longitude – float	Latitude (Degrees and Minutes DDMM.MMMM)
GPS.latitudeDegrees – float	Latitude (Decimal Degrees)
GPS.longitudeDegrees – float	Longitude (Decimal Degrees)
GPS.speed – float	Speed over Ground in Knots
GPS.altitude – float	Height in meters above MSL
GPS.year – int	Year
GPS.month – int	Month
GPS.day – int	Day of the Month
GPS.hour – int	Hour
GPS.minute – int	Minute
GPS.seconds - int	Second
GPS.satellites – int	Number of Satellites Locked
GPS.fix – bool	Have a fix T/F