Introduction

In this activity, we will use the AdafruitGPS library to communicate and control the GPS.

Materials List

1. Arduino Mega
2. Assembled Adafruit GPS Shield
3. Programming Laptop

Part 1: Direct Connect Communication

1. If not already on the Mega, put the Shield on the Mega.
2. Create a new, blank sketch in the Arduino IDE. Upload this sketch to the Mega and open the Serial Monitor.
3. Make sure the baud rate in the Serial Monitor is set to 9600 baud.
4. After uploading a blank sketch, we can flip the switch from “Soft Serial” to “Direct.” This bypasses the Arduino, allowing the Shield to communicate directly with the computer.

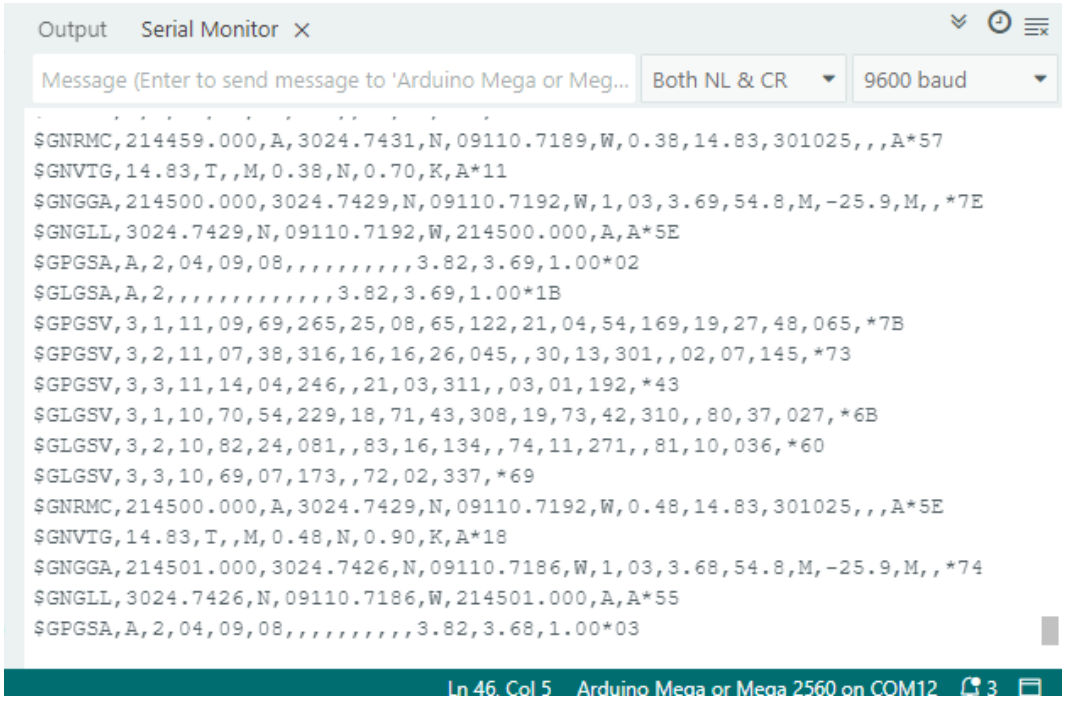

Figure 1:Here, the GPS has a fix; An older GPS that could only receive GPS signal may say GPGGA instead of GNGGA, which indicates Multiconstellation(GNSS) performance.

5. Raw GPS data should start appearing in the Serial Monitor. These are the NMEA sentences from the GPS. Figure 1 shows an example of what your Serial Monitor might look like.
   Ideally, the activity should be performed with a GPS that has a fix. If you don't have a fix, try moving the Mega and Shield closer to a window. Depending on your location, your GPS may not be able to lock onto enough satellites to get a fix. The red LED on the Shield will blink once every second if it does not have a fix, and the raw GPS data will look like Figure 2. It will blink slowly, once every 15 seconds, if it does have a fix. Even if you are in a location with no GPS reception, you will get a timestamp from the GPS.

```
$GLGSV,3,2,10,69,21,173,,82,18,096,,81,12,049,,79,07,053,*68
$GLGSV,3,3,10,74,05,258,,83,05,143,*65
$GNRMC,211617.000,V,,,,,1.32,221.15,301025,,,N*51
$GNVTG,221.15,T,,M,1.32,N,2.45,K,N*2A
$GNGGA,211618.000,,,,,0,0,,,M,,M,,*5B
$GNGLL,,,,,211618.000,V,N*69
$GPGSA,A,1,,,,,,,,,,,,,,*1E
$GLGSA,A,1,,,,,,,,,,,,,,*02
$GPGSV,3,1,10,04,69,167,25,09,63,300,16,08,55,146,18,27,52,086,*70
$GPGSV,3,2,10,16,38,040,,07,30,307,,03,10,200,,26,06,055,*73
$GPGSV,3,3,10,31,06,096,,30,05,293,*7D
$GLGSV,3,1,10,70,64,257,,80,44,011,,73,39,291,,71,34,322,*6D
$GLGSV,3,2,10,69,21,173,,82,18,096,,81,12,049,,79,07,053,*68
$GLGSV,3,3,10,74,05,258,,83,05,143,*65
```

Figure 2: Sample GPS output with no Fix, notice how most sentences have no data between the commas. In this case the GPS recently had a fix because it still lists satellites under the GSV sentences.

6. In addition to receiving the output of the GPS, we can also send to the GPS using the text box at the top of the serial monitor.
7. Before sending commands to the Shield using the Serial Monitor, make sure that "Both NL & CR" is selected next to the baud rate. Because all the PMTK commands end with a <NL><CR>, this option will automatically add them to the string.
8. Table 1 shows a list of GPS commands for changing which NMEA sentences are output. To send a command, copy it into the Serial Monitor and hit enter.
   Ensure you have copied the entire command with no space at the beginning or end.
9. Send the command to turn off all the NMEA sentences
10. After correctly sending a command, a response will be sent from the Shield "$PMTK001,314,3*36." And the NMEA sentences should stop.
11. Try sending all the commands listed in Table 1. Notice how the output from the GPS changes with each command.

Table 1: Manual GPS Commands

Table 1: GPS Commands	
Command String	**Command Result**
$PMTK314,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0*29	Turn on only the GLL sentence
$PMTK314,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0*29	Turn on only the RMC sentence
$PMTK314,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0*29	Turn on only the VTG sentence
$PMTK314,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0*29	Turn on only the GGA sentence
$PMTK314,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0*29	Turn on only the GSA sentence
$PMTK314,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0*29	Turn on only the GSV sentence
$PMTK314,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0*28	Turn on RMC and GGA sentences
$PMTK314,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0*28	Turn on all NMEA sentence output
$PMTK314,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0*28	Turn off all NMEA sentence output

12. Once you are done, turn all the outputs on by sending the Turn On All NMEA command.
13. Change the switch on the GPS to the “Soft-Serial” position.
14. If you have not already done so, connect the “Soft-Serial” pins to the pins for Serial1. This means connect pin 8 to pin 19 and pin 7 to pin 18 as shown in Figure 3.
15. At this point, you should no longer be receiving GPS data on the Serial monitor.

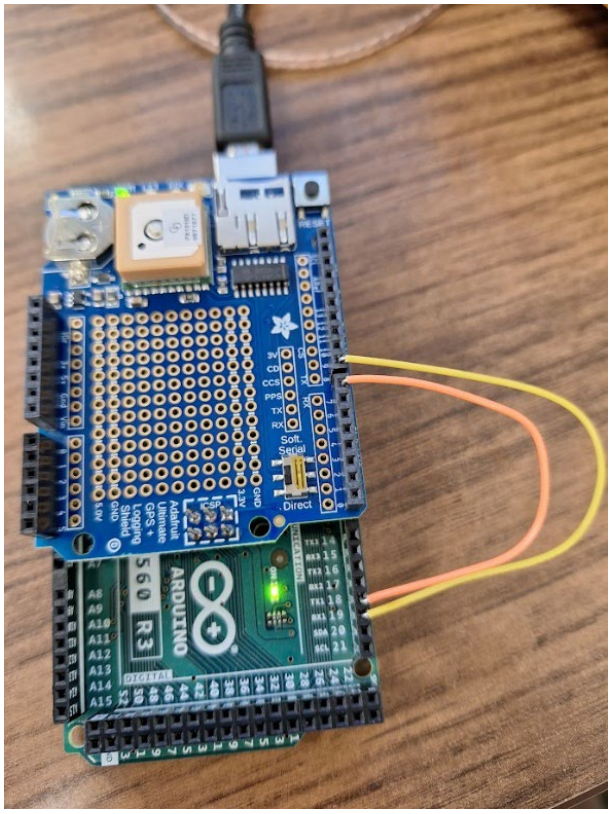

Figure 3: Jumper wiring to connect “Soft-Serial” to Serial1 pins.

Part 2: Basic GPS Read and Commanding Operations

1. Our first step will be to write a short sketch that reads the GPS and repeats it back out the Serial monitor.
   1.1. You should have previously installed the GPS library during the IDE setup. If you did not open the library manager and search for “adafruit gps” and install the library.
   1.2. Now include the library by manually typing the include statement or selecting the library from the “Include Library” option under the Sketch menu.
   Note: When adding the library via the Sketch menu, the include statements will be added, Adafruit_GPS.h, Adafruit_PMTK.h, NMEA_data.h. Only the first is needed

since it has #include statements for the other two files. You can delete the other two lines to make your code cleaner.

1.3. The IDE will not add the functions and variables to the autocomplete until it has been compiled once, so click the Verify button. Now the functions and variables from the Adafruit library should pop up in the autocomplete boxes.

1.4. Now add the code to create an Adafruit_GPS variable(don't forget the &) and initialize both the Serial and GPS using begin(). We recommend using a higher baud rate for the Serial, but leave the GPS at 9600. Your setup and globals should look Figure 4.

```
#include <Adafruit_GPS.h>

#define GPSSerial Serial1
Adafruit_GPS GPS(&GPSSerial);

void setup() {
  // put your setup code here, to run once:
  Serial.begin(38400);
  GPS.begin(9600);

}

```

Figure 4: GPS Setup code. You could just use Serial1 inside the parentheses, but the #define statement allows us to easily change the serial port for the GPS.

1.5. Now write your loop. You want to read the GPS, put that character into a variable, and print the variable to the serial monitor. Since there will not always be data from the GPS you will need an if statement to only print when a character has been read (Remember read() returns 0 when there is no data). Your loop should look like Figure 5.

```
void loop() {
  // put your main code here, to run repeatedly:
  char c;

  c=GPS.read();
  if (c){       /*Read will return a 0 there is no data
                to read, since we do not want to print those
                we use this if statement. Also note that is
                binary value 0, not the character '0' so if we
                did print it would look like anything*/
    Serial.print(c);
  }

}
```

Figure 5: GPS Readout Loop. Every time character is read from the GPS it will be

1.6. Now compile and upload your code. When you open the Serial monitor, you should see the GPS data being printed as it was during when it was in "Direct" mode.
Notice how the NMEA sentences are being printed on separate lines even though we are using print() instead of println(). That is because each sentence has the ends with the <CR><NL>, which are the characters the println() function adds to tell the serial monitor to start a new line.

1.7. If you see gibberish, ensure that you have changed the Serial Monitor baud rate to match the one specified in your setup.

1.8. If you see no data, you should check the wiring of your jumpers from pins 7 and 8 to 18 and 19. Next, check the position of the switch on the GPS; it should be in “Soft Serial”. You can place the GPS back in “Direct” mode to troubleshoot, but you will need to upload a blank sketch and reset the Serial monitor to 9600 baud. You can also use an oscilloscope to see if the GPS is transmitting by connecting it to the GPS’s TX(not the Arduino’s) pin, which should be pin 8.

2. Using Software Commands

2.1. The documentation for the command formats can be found at https://cdn-shop.adafruit.com/datasheets/PMTK_A11.pdf, but we most probably do not need to format our own command packets.

2.2. Instead, the library includes several preformatted command strings in the Adafruit_PMTK.h file. This is located inside the library folder but you can also look it up on Github for easy reference.

```
9   GPS.begin(9600);
10
11  //Commands change how often the NMEA sentences will be output
12  //NMEA updates every 1 second
13  //GPS.sendCommand(PMTK_SET_NMEA_UPDATE_1HZ);
14
15  //NMEA updates every 0.5 second
16  //GPS.sendCommand(PMTK_SET_NMEA_UPDATE_2HZ);
17
18  //NMEA updates every 5 second
19  //GPS.sendCommand(PMTK_SET_NMEA_UPDATE_200_MILLIHERTZ);
20
21  /* These commands change NMEA sentence(s) will be output
22  but each of thse commands will overwrite the previous setting
23  so the last command sent will determin the actual configurati
24
25  //Each of these sentences turn off all of the possible output
26  //GPS.sendCommand(PMTK_SET_NMEA_OUTPUT_GGAONLY);
27  //GPS.sendCommand(PMTK_SET_NMEA_OUTPUT_RMCONLY);
28  //GPS.sendCommand(PMTK_SET_NMEA_OUTPUT_GLLONLY);
29  //GPS.sendCommand(PMTK_SET_NMEA_OUTPUT_VTGONLY);
30  //GPS.sendCommand(PMTK_SET_NMEA_OUTPUT_GSAONLY);
31  //GPS.sendCommand(PMTK_SET_NMEA_OUTPUT_GSVONLY);
32
```

Figure 6: GPS Command Block, we can simply uncomment individual commands to send them to the GPS. Make sure this is after the GPS.begin() but still inside setup.

https://github.com/adafruit/Adafruit_GPS/blob/master/src/Adafruit_PMTK.h

2.3. Finally, a set of useful pre-written command statements can be downloaded in a text file from the LaACES website, next to this activity.

2.4. Download the common commands file and paste the code into your sketch inside your setup() function, but after the GPS.begin(). This allows you to send commands to the GPS by uncommenting individual lines.

2.5. Uncomment the “200 Millihertz Update” command and the “GGA Only” command. Then upload your code to the Arduino.

2.6. You should now see only GGA data and be getting new data every 5 seconds.

2.7. Now experiment with different GPS configurations, uncommenting different command lines. You will need to upload the new code

2.8. While doing so want to keep a few things in mind.

2.8.1. Some commands overwrite others; for example, if you send “RMC Only” after “GGA Only”, you will end up receiving only RMC data.

2.8.2. Once the code is uploaded, the commands will be sent by the Arduino even if there is no computer connected.

2.8.3. The coin cell battery should let the GPS retain its settings between power cycles, but it could come loose or be drained, especially in flight conditions. Therefore, it is best to assume the GPS has random settings and send any setup commands you care about during setup.

2.9. Once you are done, comment all of the commands out except the “1Hz Update” and “All Data” commands and upload the code. That should return the GPS to its previous state, where all of the data was being output every second.

Part 3: Using the Interrupt

1. Reading out the GPS with an Interrupt Service Routine ISR

1.1. Now we want to add an interrupt that will automatically try to read a character from the GPS every 1 ms.

1.2. Download the interrupt code from the LaACES website next to this activity and add it to your sketch. You should place this above your setup() and loop() functions, like Figure 5.

1.3. This adds two functions: the ISR will read out the GPS, and the useInterrupt() function turns the ISR on and off.

1.4. This code alone does not enable the interrupt; we need to call useInterrupt() to turn it on. The natural place to do so is at the end of the setup.

```
4   Adafruit_GPS GPS(&GPSSerial);
5   /*The following code can be added to you sketch to allow the
6   the Timer interrupt to read a GPS character ~1 millisecond*/
7
8   boolean usingInterrupt = false;
9   /* This variable that is changed when the interrupt is turned
10  It can be used to check if there is an interrupt is on or off
11
12  ISR(TIMER0_COMPA_vect) {
13  /******* Interrupt Service Routine ************************
14   *  This is the interrupt service routine. It reads a charact
15   ***********************************************************
16    GPS.read();      // Read a character from the GPS
17  }
18
19  void useInterrupt(boolean v) {
20  /******* useInterrupt *************************************
21    This function is enables or disables the Timer interrupt.
22    So
23   ***********************************************************
24    /* millis() uses Timer0, so we'll interrupt at the half mil
25    that prevents interference with millis which happens at the
26    if (v) {
27      OCR0A = 0xAF;    //This line controls when the trigger ha
28      TIMSK0 |= _BV(OCIE0A); //This enables the interrupt by ch
29      usingInterrupt = true;
30    }
31
32    // Do not call the interrupt function COMPA anymore
33    else {
34      TIMSK0 &= ~_BV(OCIE0A);  //This enables the interrupt by
35      usingInterrupt = false;
36    }
37  }
38
39  void setup() {
40    // put your setup code here, to run once:
```

Figure 7: Interrupt readout code. Notice we have added it after the GPS variable declaration, but before setup.

1.5. Add a line to your setup to turn the interrupt on in your setup as shown in Figure 9.

1.6. Now we need to write the loop.

1.7. Since the interrupt is automatically reading the GPS, all we need to do is check if a new NMEA sentence has been received and print it once it has.

1.8. Add the code to your loop to do this. An example is shown in Figure 8.

1.9. Upload your code, and you should see all of the NMEA sentences printed out much as before. While this looks similar, the important difference is that we do not need to constantly call read() in the loop, which allows us to perform other steps in the loop. The interrupt will automatically check for GPS data.

```
72
73    //This commands turn on all outputs that the GPS supports
74    GPS.sendCommand(PMTK_SET_NMEA_OUTPUT_ALLDATA);
75    //This command turns off all of the NMEA sentences
76    //GPS.sendCommand(PMTK_SET_NMEA_OUTPUT_OFF);
77    useInterrupt(true);
78  }
79
```

Figure 9: Enabling the interrupt by calling useInterrupt(true); This is done at the end of the setup() function after all the GPS commands have been sent.

```
void loop() {
  // put your main code here, to run repeatedly:
  if (GPS.newNMEAreceived()){
    Serial.print(GPS.lastNMEA());
  }

}
```

Figure 8: GPS readout loop using the interrupt and the newNMEAreceived() function. Every time the interrupt finishes a sentence the if statement is true and the NMEA is printed. Because lastNMEA() changes newNMEAreceived() back to false, each sentence is only printed out once.

2. Impact of large delays.

2.1. Now, let's see the impact of a long delay. A delay(3300); line to the bottom of your loop(). Ensure it is outside the if statement for printing.

2.2. Upload the code, and you should see the output similar to Figure 10. We receive a complete NMEA sentence approximately every 3 seconds, but which one we receive is random. Since the sentences are complete, we can tell we are not missing characters. However, since we only check for a sentence every few seconds, we only get the most recent one that happens to be available. This is

```
15:44:35.401 -> $GLGSV,2,2,08,72,23,329,,80,16,042,,82,09,059,,83,09,10
15:44:38.733 -> $GNVTG,240.43,T,,M,0.12,N,0.22,K,D*24
15:44:42.003 -> $GNGLL,3024.7529,N,09110.7068,W,204440.000,A,D*5A
15:44:45.329 -> $GPGSV,3,3,09,31,13,086,19*46
15:44:48.614 -> $GNVTG,240.43,T,,M,0.16,N,0.30,K,D*23
15:44:51.939 -> $GNVTG,240.43,T,,M,0.08,N,0.15,K,D*2B
15:44:55.257 -> $GPGSV,3,1,09,04,84,162,27,09,53,314,16,27,50,107,17,16
15:44:58.548 -> $GNRMC,204456.000,A,3024.7475,N,09110.7012,W,0.23,240.4
15:45:01.815 -> $GNVTG,240.43,T,,M,0.15,N,0.28,K,D*29
15:45:05.145 -> $GLGSA,A,3,,,,,,,,,,,,,2.95,2.79,0.97*12
```

Figure 10: GPS when there is a long delay in between calls of newNMEAreceived().

critical because each sentence has different data. So we may have a long gap before we get an updated altitude or latitude.

2.3. The simplest solution to this problem is to just turn off the data we don't care about. Go to your setup and comment out the "All Data" command and uncomment the "GGA Only" command. Now upload your code.

```
15:47:49.420 -> $GNGGA,204747.000,3024.7490,N,09110.7267,W,2,06,1.60,32.5,M,-25.9,M,,*75
15:47:52.693 -> $GNGGA,204750.000,3024.7498,N,09110.7264,W,2,06,1.61,32.8,M,-25.9,M,,*74
15:47:55.986 -> $GNGGA,204753.000,3024.7504,N,09110.7246,W,2,06,1.61,33.2,M,-25.9,M,,*78
15:47:59.295 -> $GNGGA,204757.000,3024.7482,N,09110.7239,W,2,07,1.61,33.8,M,-25.9,M,,*70
15:48:02.608 -> $GNGGA,204800.000,3024.7471,N,09110.7239,W,2,07,1.61,34.2,M,-25.9,M,,*7C
15:48:05.910 -> $GNGGA,204803.000,3024.7482,N,09110.7233,W,2,05,2.09,34.6,M,-25.9,M,,*72
15:48:09.209 -> $GNGGA,204807.000,3024.7455,N,09110.7229,W,2,05,2.09,35.1,M,-25.9,M,,*71
```

Figure 11: GPS readout with a long delay, but only a single sentence enabled. Since only

2.4. You should now see an output like Figure 11. Since GGA data is the only data being transmitted, we will always have up-to-date data.

2.5. Comment out the delay() in your loop and turn all of the outputs on by commenting out the "GGA Only" command and uncommenting the "All Data" command. This should return the code to print all data out every second.

Part 4: Parsing the NMEA Sentences

1. Using the parse() function

1.1. The NMEA text strings can be cumbersome to deal with, and often we only care about a piece of the data, such as the time or altitude. So we can use the parse function and the GPS internal variable to access specific data.

1.2. Modify your loop to call GPS.parse() on the lastNMEA() instead of printing it.

1.3. Then, outside of the if statement, print out the time in Hours:Minutes:Seconds format using GPS.hour, GPS.minute, and GPS.seconds. Your loop should look like Figure 12.

```
void loop() {
  // put your main code here, to run repeatedly:
  if (GPS.newNMEAreceived()){
    GPS.parse(GPS.lastNMEA());
  }
  Serial.print(GPS.hour);
  Serial.print(":");
  Serial.print(GPS.minute);
  Serial.print(":");
  Serial.println(GPS.seconds);

  //delay(3300);
}
```

Figure 12: Parsing the NMEA sentence and then printing the GPS time out Hours:Minutes:Seconds format

1.4. Upload your code, and you should see the time rapidly being printed to the Serial monitor. Even if you do not have a fix, you should see the time counting up. If you do have a fix, the time should be accurate; just remember it will be 5 or 6 hours ahead since it is the GMT(Greenwich Mean Time) time zone instead of Central.

1.5. Now, try adding your delay back in by uncommenting that line and reuploading the code. You should see something like the data in Figure 13 where it takes a while for the

time to even from the initial value of 0, and then will repeat time and only occasionally update.

1.6. This is due to the random NMEA sentence effect we saw earlier. Since we only occasionally get an NMEA sentence with time, we will only occasionally update GPS.hour, GPS.minute, and GPS.seconds.

```
80  void loop() {
81    // put your main code here, to run repeatedly:
82    if (GPS.newNMEAreceived()){
83      GPS.parse(GPS.lastNMEA());
84    }
85    Serial.print(GPS.hour);
86    Serial.print(":");
87    Serial.print(GPS.minute);
88    Serial.print(":");
89    Serial.println(GPS.seconds);
90
91    delay(3300);
92  }
93
```

Serial Monitor × Output

Message (Enter to send message to 'Arduino Mega or Mega 2560' on 'COM12')

```
0:0:0
0:0:0
0:0:0
0:0:0
0:0:0
0:0:0
21:17:56
21:17:56
21:17:56
21:17:56
21:17:56
21:17:56
```

Figure 13: Attempting to use a delay results in the time not updating and skipping occasionally because the parsed sentence does not have a timestamp,

Part 5: Using Timer instead of delay()

1. So, how can we slow down our output record data at a slower rate, while making sure our data is accurate and up to date? Being able to do so will be especially important in flight conditions where data storage is limited and we do not want to process 500 points of data every second.

1.1. Instead of using the delay() function, which just stops the program, we can use the Arduino's timers and an if statement to only print data when a certain amount of time has passed since the last time we printed something.

1.2. The millis() function is a built-in function that returns the number of milliseconds that have elapsed since the program began execution. We can use this to time our print statements so we only print every 5 seconds (or other interval) while still calling parse() often enough to update all of the data.

```
1  #include <Adafruit_GPS.h>
2
3  #define GPSSerial Serial1
4  Adafruit_GPS GPS(&GPSSerial);
5  unsigned long timer_5s;
6
7  /*The following code can be added
```

Figure 14: Timer variable declaration. An unsigned long will roll over at ~4 billion (or after 4 million seconds if we are using it to count milliseconds), making sure we do not have any timer overflows.

1.3. First, let's create a variable to store the time. We want to make it global so it doesn't get reset each time through the loop. We also do not want to use a regular int because that will roll over to negative at ~30,000 (so after 30 seconds). Create an unsigned long variable at the top of your program, as shown in Figure 14.

1.4. To start timer, we need to record the current time(output of millis()). Let's do that at end of the setup function as Figure 15.

1.5. Now, every time through the loop, we check for a NMEA sentence and parse it if one is available.
1.6. We also want to check if it has been longer than 5 seconds since the last time we printed the time out. If so, print out the time and restart our 5-second timer.
1.7. Add an if statement to your loop to accomplish that, which should be similar to Figure 16.
1.8. When you upload the code and look at the Serial Monitor, you should see an output like Figure 17 with the time being properly updated every time it is printed out.
1.9. As a final addition, add a second timer to print out the Date in Month/Day/Year format every 10 seconds. You will need to use the GPS variables GPS.month, GPS.day, and GPS.year.

```
76    GPS.sendCommand(PMTK_SET_NMEA_
77    //This command turns off all c
78    //GPS.sendCommand(PMTK_SET_NME
79    useInterrupt(true);
80    timer_5s=millis();
81  }
82
83  void loop() {
```

Figure 15: Starting the timer by recording the value of millis() at the end of the setup()

```
void loop() {
  // put your main code here, to ru
  if (GPS.newNMEAreceived()){
    GPS.parse(GPS.lastNMEA());
  }
  if ((millis()-timer_5s)>5000)
    {Serial.print(GPS.hour);
    Serial.print(":");
    Serial.print(GPS.minute);
    Serial.print(":");
    Serial.println(GPS.seconds);
    timer_5s=millis();
    }

}
```

Figure 16: If statement to print out the current time, and then restart the 5-second count by saving the current millis() in timer_5s.

```
22:16:22
22:16:27
22:16:32
22:16:37
22:16:42
22:16:47
22:16:52
22:16:57
22:17:2
22:17:7
```

Figure 17: Time output using an if statement. Notice how the time is always updated, even though we are waiting 5 seconds, and all of the NMEA sentences are active.