Introduction

In this activity, students will learn how to interface with the microSD card using the Adafruit Ultimate GPS Logger Shield. This version is designed to be used for the shield revision with the 6 6-pin ICSP Header on the GPS shield and the current version of the SD library(v.1.3.0 at time of writing) from the Arduino IDE manager. Because of the length of the SD card programming activity is split into two parts.

Materials List

1. Programming laptop
2. Arduino Mega
3. Assembled Adafruit GPS Shield
4. MicroSD card (32 GB or smaller)
5. USB SD card reader (if required)

Basic SD File Writes

1. Initializing the SD Card
   1.1. Our step will be to add the library to our sketch by including SD.h. Just like the GPS library, we can either manually type the include or use the menu option under the **Sketch** menu.
   1.2. Note: Even though the SD library uses the SPI library, you do not need to have an include statement for SPI.h. The SD Library will automatically include SPI.
   1.3. Just like the GPS library, once you add the library, compile the sketch by hitting the Verify/Check button. This will add the library contents to the IDE's autocomplete.
   1.4. Now add a define statement for the Chips Select pin (which is pin 10).
   1.5. The SD card variable is already defined inside the library, so we do not need a declaration like we did with the GPS. Instead, we just

```
#include <SD.h>

#define CS_pin 10

void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
  Serial.println("Initializing SD card...");
  if (!(SD.begin(CS_pin))){   //Attempt to initialize the SD card, if su
    Serial.println("Initialization Failed"); // the ! is a NOT operator
    while(1); //Stops the code here since
  }
  Serial.println("Initialization successful");
}
```

Figure 1: SD Card Setup. If the initialization fails the code gets the to the while(1); loop and stops since 1 will never be false.

need to call the begin() function to tell the library which chip select pin.

1.6. We can use the return value of the SD.begin() function to determine if the SD card was actually detected. So by adding an if statement as shown in Figure 1.

```
Initializing SD card...
Initialization Failed
Initializing SD card...
Initialization successful
```

Figure 2: Message from failure and the successful SD card initialization.

1.7. Without the SD card installed, upload the code to your Arduino. Since there is no card installed when you open the Serial monitor, you should see the messages for the Arduino attempting to initialize the SD card and failing.

1.8. Now insert the SD card(you should hear a click) and hit the reset button on the GPS shield. This will restart the code on the Arduino. You should now see that the Arduino has successfully initialized the SD card.

2. Basic file write operations

2.1. To write data, we need to open a file, write the data and then close the file. Lets add code to the loop to do so.

2.2. To open the file for writing, we call the SD.open() function and pass it the name of the file and the constant FILE_WRITE to open the file in write mode. This will return a File object that we can use for writing and closing, so we also need to create a File variable to save the returned object.

2.3. Declare a variable called myFile using the File type and then open a file called datafile.txt using the open() function. As shown in

2.4. If the file failed to open for some reason, myFile will evaluate to 0 so we can use an if statement and some Serial.println() to check if the file successfully opens. An example of this is shown in Figure 3.

```
5
6   void loop() {
7     // put your main code here, to run repeatedly:
8     File myFile;
9     myFile = SD.open("datafile.txt", FILE_WRITE);
0     if (myFile){
1       Serial.println("File opened");
2     }
3     else{
4       Serial.println("Failed to open file");
5     }
6     delay(1000); //delay slow the file down
7   }
8
```

Figure 3: Simple open a file on the SD card.

2.5. Upload your code to the Arduino with the SD.card installed. You should see Arduino successfully open the file.

2.6. If you eject the SD card, you should see the Serial message change to your Failure message, which makes sense since the SD card is removed.

2.7. If you look at the SD card, you will see there is a single file, DATAFILE.TXT. If you open the file, you will see that it is empty, which makes sense since we have not actually written any data. Also, notice that the library has capitalized all of the letters in the file name.

2.8. Now modify the name of the file to "toolongdatafile.txt". Reinstall your SD card and upload the code. You will see that the open function now fails. The name we have chosen for the file is invalid; the library restricts the file name to the 8.3 format.

2.9. Change your file name back to a valid format like datafile.txt.

2.10. To write data, just need to call the print() or println() functions on the opened file. Add a myFile.print() line to write a single character like "a" to the file and a Serial message to . Then also increase your delay to 10000 (10 seconds) as shown Figure 5.

2.11. Upload your code to the Arduino with the SD card installed. Once you see the "Wrote data" message, eject your SD card and look at the data file. It should still be empty. What happened?

2.12. By default, the library waits until the SPI transmit buffer is full to send the data to the SD card, so if

```
16  void loop() {
17    // put your main code here, to run repeatedly:
18    File myFile;
19    myFile = SD.open("toolongdatafile.txt", FILE_WRITE);
20    if (myFile){
21      Serial.println("File opened");
22    }
23    else{
```

Output Serial Monitor ×

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

```
Failed to open file
```

Figure 4: Example of an invalid filename causing the library to fail at opening the file.

```
void loop() {
  // put your main code here, to run repeatedly:
  File myFile;
  myFile = SD.open("datafile.txt", FILE_WRITE);
  if (myFile){
    Serial.println("File opened");
    myFile.print("a");
    Serial.println("Wrote Data");
  }
  else{
    Serial.println("Failed to open file");
  }
  delay(10000); //delay slow the file down
}
```

Figure 5: Loop code to write the character "a" to the file.

```
// put your main code here, to run repeatedly:
File myFile;
myFile = SD.open("datafile.txt", FILE_WRITE);
if (myFile){
  Serial.println("File opened");
  myFile.print("a");
  myFile.flush();
  Serial.println("Wrote Data");
}
else{
  Serial.println("Failed to open file");
}
delay(10000); //delay slow the file down
}
```

Figure 6: Loop code modified to call flush after every write, this should now send the data to SD immediately,

there are large delays and our data is small, we run the risk of losing data if there is a power outage. The flush() function forces any waiting data to be sent to the SD card immediately.

2.13. Add a line to call flush() function immediately after your print, as shown in Figure 6. Now, when you upload your code and let it run, it should write data into the file. You do not need to call flush() after every write, but it is good practice to call it occasionally unless you are writing a large amount of data.

2.14. Now reduce your delay to 1000 (for 1 second) and upload your code. With the SD card installed, you should see the yellow L13 LED blink every 1 sec. This LED is connected to the clock line for the SD card, so you can see it light up every time the Arduino talks to the SD card. This can be a useful check when troubleshooting. If the LED is not blinking, you are not sending any data to the SD card.

2.15. If you look at your data file, you should just see a single line of “a” characters. Usually, writing out data to a single line is not a useful format. Let's change our program to be slightly more realistic.

2.16. First, make your File variable global by moving it outside the loop(). Also, make the file name a String variable called filename. Remember, the globals are usually above the setup, after the libraries, as shown Figure 8. Making these

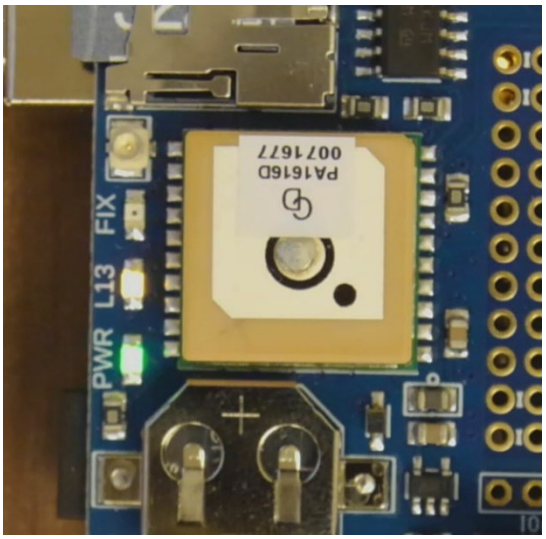

Figure 7: The L13 LED on the shield lights when the Arduino is communicating to the SD card.

```
#include <SD.h>

#define CS_pin 10

File myFile; //Global file variable
String filename = "datafile.txt";

void setup() {
```

Figure 8: Global variable declarations for myFile and filename.

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

  myFile = SD.open(filename, FILE_WRITE);
  if (myFile){
    Serial.println(filename+" opened");
    myFile.println("Hello World");
    myFile.println("Data Packet 1");
    myFile.println("Data Packet 2");
    myFile.close();
    Serial.println("Wrote Data");
  }
  else{
    Serial.println("Failed to open file");
  }
  delay(1000); //delay slow the file down
}
```

Figure 9: A more realistic version of a loop to write data to the SD card. Here, we open the file at the top of the loop, write data in multiple lines, and close the file when finished writing.

global will allow them to be used everywhere in the program, in setup(), loop(), or any functions we add.

2.17. Now modify your loop so that it uses the filename variable to open the file.

2.18. We can also change the Serial.println() to tell us which file was opened. Since filename is a String, we can use the + operator to combine it with other strings, as shown in Figure 9.

2.19. Inside the if statement, change the print() to instead write three different strings on separate lines.

```
1   aaaaaaaaaaaaaaaaa
2   Data Packet 1
3   Data Packet 2
4   Hello World
5   Data Packet 1
6   Data Packet 2
7   Hello World
8   Data Packet 1
9   Data Packet 2
10  Hello World
11  Data Packet 1
```

Figure 10: Sample data file output

2.20. Finally, it is usually good practice to close the file when we are done writing data to it. This both flushes data to be written and removes the link between the myFile variable and the file on the SD card. That forces us to use open() the next time we write data, so it is not ambiguous which file the data is being written to.

2.21. Upload your modified code and let it run with the SD card installed. Once you have recorded a few data points, eject the SD card. When you open the data file, you should see data similar to Figure 9.

Note: This activity continues with Activity 4.08 SD Card Programming Pt II.