Summary:

This activity will walk students through how to save data onto an SD card using the Adafruit Ultimate GPS Logger Shield. By the end of this activity, students will understand how to save data to an SD card, read data from an SD card, and create an csv file. This continues from part 1.

Materials:

Each student should have the following materials, equipment and supplies:

- Computer with Arduino IDE installed and microSD or SD reader
- USB-AB programming cable
- Arduino Mega microcontroller with assembled Adafruit Ultimate GPS Logger Shield attached
- microSD card
- microSD to SD adapter (if computer has an SD reader instead of a microSD reader)

Procedure:

Activity D: Creating a CSV

1. Now we want to modify our code so it creates a simple data packet consisting of 4 numbers and will write those to a .csv file

2. Where you created your global variables change the filename of the created file to "csvtest.csv" and create an array (with 4 entries) of floats. Initialize these values to {1.5, 3.25, 5.0, 7.57}. This array of floats will act like our data.

```
/* SD Card & Files ************************************************************
#define CS 10                  // SPI chip select pin
#define MOSI 11                // SPI master out, slave in pin
#define MISO 12                // SPI master in, slave out pin
#define CLK 13                 // SPI clock pin
File myFile;                   // File object
char fileName[21] = "test.csv"; // Name of file to be created on SD card

/* Other global variables *****************************************************
float testArray[4] = {1.5,3.25,5.0,7.57};
```

Figure 1: Here is our new declaration for our filename, notice we have changed the extension to "csv". And thee we can see a float array we have declared with initial values that will simulate data.

3. Edit setup() so that it only begins communications with the Serial Monitor and the SD card. Remove the code that previously wrote data to the file. Then close the file object.

4. In loop(), have the follow actions performed in order: open the SD card file to write; write these 4 values from our data array, separated by commas, to the SD card on a new line; flush; close the SD file; then add 0.5 to all values of the array to change the values; print "Done!" to the Serial Monitor; and wait for 3 seconds.

```
void setup() {

  /* Start communications with peripherals ***********************************************/
  Serial.begin(9600);                              // Communicate with Serial Monitor

  Serial.println("Initializing SD card...");
  if (!SD.begin(CS, MOSI, MISO, CLK)) {            // Initialize communication with SD card
    Serial.println("Initialization failed :(");    // If initialization failed, print it
    while(1);                                      //    and enter a forever while loop
  }
  Serial.println("Initialization successful!");    // If initialization successful, print it
  myFile = SD.open(fileName, FILE_WRITE);

  if (myFile) {
    Serial.println("Open successful for " + String(fileName));
    myFile.flush();
  }
  else Serial.println("There was an error opening " + String(fileName));
  myFile.close();
}
```

Figure 2: The modified setup function. Notice how the setup not just initialized the SD and Serial objects are creates out data file by opening it but does not write any data to it.

a. First we need open the file using SD.open(). Then we use an if statement to verify that the file was successfully opened, if there was an error it will skip over the portion of the code that does the file writes.

   Next, we use a for loop to write the individual values of the of the array and their comma separators one at a time. Notice the way this loop is written it ill have comma at the end of the line. We then print an empty string with println() to start a new line and call flush to ensure the write happens. We have an else statement to give us an error message in the event of the file not correctly opening.

   We then close the file and send our done message, letting us know the write is complete.

```
void loop() {
  /* Open, write to, and close an SD file **********************************
  myFile = SD.open(fileName, FILE_WRITE);

  if (myFile) {
    Serial.println("Open successful for " + String(fileName));

    for (int i=0; i<4; i++) {
      myFile.print(testArray[i]);
      myFile.print(",");
    }
    myFile.println("");
    myFile.flush();
  }
  else Serial.println("There was an error opening " + String(fileName));

  myFile.close();
  Serial.println("done");
```

Figure 3: Shown is how to write all the values from a length 4 array with commas separating them. The println call after the for loop starts a new line in the file.

b. The remaining part of the loop is simple. We use another 4 loop to add 0.5 to each of our data numbers to simulate a changing value and then a delay to wait for 3 seconds.

```
/* Add .5 to all values of array and delay for 3 seconds ***
for (int i = 0; i<4; i++) testArray[i] = testArray[i] + .5;
delay(3000);
```

Figure 4: We increment all our data values and wait 3 seconds after which the loop starts over and writes our new values to the file.

5. Upload the sketch. Wait until "Done!" has printed to the Serial Monitor at least 5 times and then remove the SD card.

6. Open the CSV file in Excel as well. It should look like Figure 5. Having the CSV format makes looking at data and analyzing data in Excel much easier because Excel separates it for the user. You can also open the file in a text editor and you will see the raw number and commas. One import thing to note in the raw output you will see that all of the numbers are rounded to decimal places, excel will automatically drop any trailing zeros.

	A	B	C	D
1	1.5	3.25	5	7.57
2	2	3.75	5.5	8.07
3	2.5	4.25	6	8.57
4	3	4.75	6.5	9.07
5	1.5	3.25	5	7.57
6	2	3.75	5.5	8.07
7	2.5	4.25	6	8.57
8	3	4.75	6.5	9.07
9	3.5	5.25	7	9.57

Figure 5: This is what the csv file should look like when opened in Excel. Excel does the parsing automatically, so the data is easy to look

The print function will default to 2 decimal places when converting the float to a series of characters. We can change that by passing an argument after then number to print our like

```
myFile.print(testArray[i],4);
```

Figure 6: Forcing the printed number to be to 4 decimal places.

shown below. This is something you will want to keep in mind when deciding how to write your payload data. Try modifying your code to change the number of decimals for you data.

7. We probably want to make one more modification to our program, we want to add a header to the file that will give the titles to our columns. To do this we need to go back and modify our setup() so we write the header line immediately after we open our file.

```
if (myFile) {
  Serial.println("Open successful for " + String(fileName));

  myFile.println("Num1,Num2,Num3,Num4");       //Add a header to the top of the file
  myFile.flush();
}
```

Figure 7: Adding a header for column labels to our data file.

So if we delete our old data file, reinstall our SD card and upload our modified code we will get a .csv that should automatically add labels to the top our columns if we open it in excel.

Try resetting the Arduino and see what happens, you should see the data just get added to the end of the file. This is a concern especially since if we had broken data in to several files it would return to the first file on loss of power.

Activity F: Breaking up Data to multiple files

	A	B	C	D
1	Num1	Num2	Num3	Num4
2	1.5	3.25	5	7.57
3	2	3.75	5.5	8.07
4	2.5	4.25	6	8.57
5	3	4.75	6.5	9.07
6	3.5	5.25	7	9.57

Figure 8: Our new data file with column headers.

Now we want to work on a more realistic example. We are going to break have our data broken into multiple data files. We will give each of the files a unique number identifying number. And we will make it so that the when the Arduino power is cycled that the data will resume writing the data to a new data file.

1. In our global variables we need to give our file name a different structure. We will name our files with DR#.csv with the number the counting as we create new files. We will start with a default name of "DR0.csv"/

   We also need a temporary variable for that we will use for the manipulating the file name. The actual file name needs to a char array for the SD card library. But the String type is easier to manipulate so that's why we use the two different variables.

   We also need an integer to track the current file number.

```
char fileName[50]="DR0.csv";  //The character arrary with the file name, initialized to the first name to try
String this_file;             //A String that keeps the current file name
                              //We have both of these because the it is easier to manipulate the string combine
                              //the parts that make up our file name but need to give a character array for the filename

int filenum=0;                //A varible for tracking which file we are on ie DR0, DR1, ...
```

Figure 9: The new global variables for creating the file names.

2. We want to add 2 more columns to our data lines that allow us to uniquely identify each data line, the file number and the data line number within the file. To accommodate that we want to add those labels to our file header.

```
String file_header="FileNum,ID,Num1,Num2,Num3,Num4";
float testArray[4] = {1.5,3.25,5.0,7.57};    //Our initial se
```

Figure 10: Our updated header with our 2 new ID fields.
Notice the actual data remains unchanged

3. Next, we need two additional global variables we need to the maximum number of data records per file and the current number of data records written to a file.

```
int max_writes=10;      //This variable will be the maximum number of data lines written to a file
int num_writes;         //This variable will track the current number of data lines written to a file
```

Figure 11: max_writes is the maximum number of data packets written to single file and num_writes is where we will keep track of the current number of data packets that have been written to the current file.

4. Finally, we want to move our file creation and file writing in to separate functions so that we can just call those files. In general, it is best to break individual tasks to individual functions for ease of use and organization.

```
void new_file();            //The function for creating a new file,
                            //this will check the for the next available file and create it
void write_data();          //The function where we will actually write the data
```

Figure 12: The new functions we will use for writing and creating files.

5. Our setup() function is going to be similar except we now just want to call our new new_file() function to actually create our new file.

```
void setup() {
  /* Begin serial communications ***************************************/
  Serial.begin(9600);   // Begin communication with the computer


  Serial.println("Initializing SD card...");
  if (!SD.begin(CS, MOSI, MISO, CLK)) {           // Initialize communication with SD card
    Serial.println("Initialization failed :(");   // If initialization failed, print it
    while(1);                                     //  and stop the program here
  }
  Serial.println("Initialization successful!");   // If initialization successful, print it

  new_file();     //Open our data file, this will either start at DR0 or resume at the next file
}
```

Figure 13: The new setup() function simplified by the use of the new file function().

6. Since we have moved our file creation and writing in to separate functions our loop() is super simple. We just need to check if we have reached the max number of data points in a file and create a new if so. And then actually write the data to the current file.

   We also have a small delay just to limit the number of writes that occur.

```
void loop() {
    if (num_writes > max_writes) new_file();
    //Once we have written the max number of datapoint start a new file
    write_data(); //This will write our data to the file
    delay(500); //add a short delay between writes
}
```

Figure 13: Our loop(), again very simple because the work is being done by the two functions.

7. Our write data function is mostly similar to our old loop() function. The first addition we need to make is that we want to write the current number. Next we want to write current datapoint number. We use the two for loops to write the data and increase the numbers as before. And finally we increment the number of data point in the current file.

```
void write_data(){
    myFile = SD.open(fileName, FILE_WRITE);  //Open the file
    myFile.print(filenum);  //Print the file number to the record
    myFile.print(",");
    myFile.print(num_writes); //Print the write number
    myFile.print(",");
    /* Print out our set of data */
    for (int i=0; i<4; i++) {
      myFile.print(testArray[i]);
      myFile.print(",");
    }
    myFile.println(); //Start a new line
    myFile.flush();         //Forces the write
    myFile.close();         //Close the file
    for (int i = 0; i<4; i++) testArray[i] = testArray[i] + .5;  //Increments our data values
    num_writes+=1;          //Increment the number of writes

}
```

Figure 14: The function to actually write our data, notice that it is very similar to our previous loop() with a few additions.

8. We want our new_file() function to check the SD card and find the next unused file name and use that for the new file. To do that we will use the SD.exists() function. This will return True when the file already exists and false when it does not.

   So we use a while loop that first tries the existing file name, this way on loss of power it will try DR0.csv first. If the SD card is blank it will create that file and proceed.

   If there are old files on SD card it will enter the loop. Each time it goes through the loop it will increase the current filenumber (which matches the number in the filename) and combine that number with "DR" and ".csv" to create a new file name. Then it returns to the top of the loop and checks and see if the file already exists.

```
void new_file(){
    while (SD.exists(fileName)){          //This loop will try execute if the file name already exists, in normal operation this loop
                                          //execute once, incrementing to the next file. But on a restart of power this will check all the
                                          //existing files and resume the
      Serial.print(fileName);             //This prints out the old file name
      filenum += 1;                       //Increments the filenum counter by 1
      this_file=String("DR"+String(filenum)+".csv");  //This creates creates a temporary string with the new file number
      //We cannot simply use the + to combine the DR, filenum, and .csv with the char array so we do this with the String type
      //DR1.csv, DR2.csv, DR3.csv ...
      this_file.toCharArray(fileName, 50);    //This copies the string this_file into the Char array fileName
      Serial.println(" already exists trying "+this_file);  //Prints out the new file name it is about to try
    }
```

Figure 15: Here we loop through possible file names until an unused one is determined.

This will happen until an unused file name is found. Once that occurs it will create the file and write the header. This function will both increment a new file after every file reaches the max number of data points and correctly resume with the next highest available file number after a restart.

```
myFile = SD.open(fileName, FILE_WRITE);      //Once a nonexisting name is found open the file
Serial.print("File created: ");
Serial.println(fileName);
myFile.println(file_header);      //Writes the first header line to our new file
myFile.flush();
myFile.close();
num_writes=0;
```

Figure 16: Finally we open the new file and update our data writes to 0.

9. We also need to reset the number of data points back to zero after every time this happens.

Activity F: Combining GPS and SD Functionality

Note: NMEA Sentences are very nice for CSV files because their information is already comma separated. The purpose of this activity is to save NMEA sentences to an SD file.

1. If not already, jumper the Shield to the Mega to allow for Serial communication for the GPS module. Create a new sketch called *NMEA_SD.ino*. Clear the SD card.

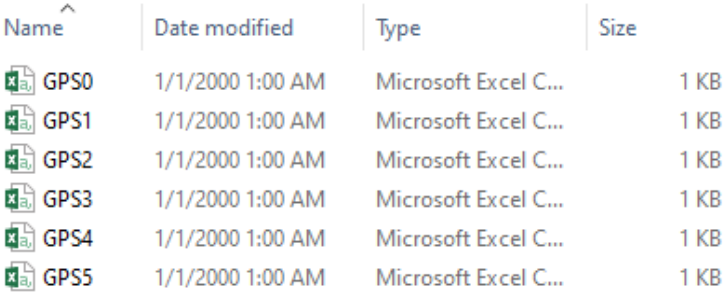
2. Using previous activities or sketches as a reference, set up communication with GPS and the SD card. Have the GPS send RMC only sentences. And have it output at a 1 Hz rate.

3. Do not forget to include the necessary interrupt and GPS reading functions from the previous activities.

4. When a new NMEA sentence is received, copy that sentence to a variable and just write it to the SD card. Don't forget to flush after writing. Since only a single sentence is being sent this should write a sentence to a single file.

5. Set the program to write 20 sentences to a single file and create a new file after that. You should be able to modify the previous

6. Upload the sketch and have it create at least 4 files before removing the SD card.Look at the files created on the computer, it should look like Figure 17



Name	Date modified	Type	Size
GPS0	1/1/2000 1:00 AM	Microsoft Excel C...	1 KB
GPS1	1/1/2000 1:00 AM	Microsoft Excel C...	1 KB
GPS2	1/1/2000 1:00 AM	Microsoft Excel C...	1 KB
GPS3	1/1/2000 1:00 AM	Microsoft Excel C...	1 KB
GPS4	1/1/2000 1:00 AM	Microsoft Excel C...	1 KB
GPS5	1/1/2000 1:00 AM	Microsoft Excel C...	1 KB

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	$GPRMC	230017.402	V					0	0	61119			N*42
2	$GPRMC	230018.402	V					0	0	61119			N*4D
3	$GPRMC	230019.402	V					0	0	61119			N*4C
4	$GPRMC	230020.402	V					0	0	61119			N*46
5	$GPRMC	230021.402	V					0	0	61119			N*47
6	$GPRMC	230022.402	V					0	0	61119			N*44
7	$GPRMC	230023.402	V					0	0	61119			N*45
8	$GPRMC	230024.402	V					0	0	61119			N*42
9	$GPRMC	230025.402	V					0	0	61119			N*43
10	$GPRMC	230026.402	V					0	0	61119			N*40
11	$GPRMC	230027.402	V					0	0	61119			N*41
12	$GPRMC	230028.402	V					0	0	61119			N*4E
13	$GPRMC	230029.402	V					0	0	61119			N*4F
14	$GPRMC	230030.402	V					0	0	61119			N*47
15	$GPRMC	230031.402	V					0	0	61119			N*46
16	$GPRMC	230032.402	V					0	0	61119			N*45
17	$GPRMC	230033.402	V					0	0	61119			N*44
18	$GPRMC	230034.402	V					0	0	61119			N*43
19	$GPRMC	230035.402	V					0	0	61119			N*42
20	$GPRMC	230036.402	V					0	0	61119			N*41

Figure 17: Left – This is what the files on the SD card should look like. Right – This is what an individual file on the SD card should look like when it's opened on Excel. Again, the GPS did not have a fix for this run, so some columns are empty.