

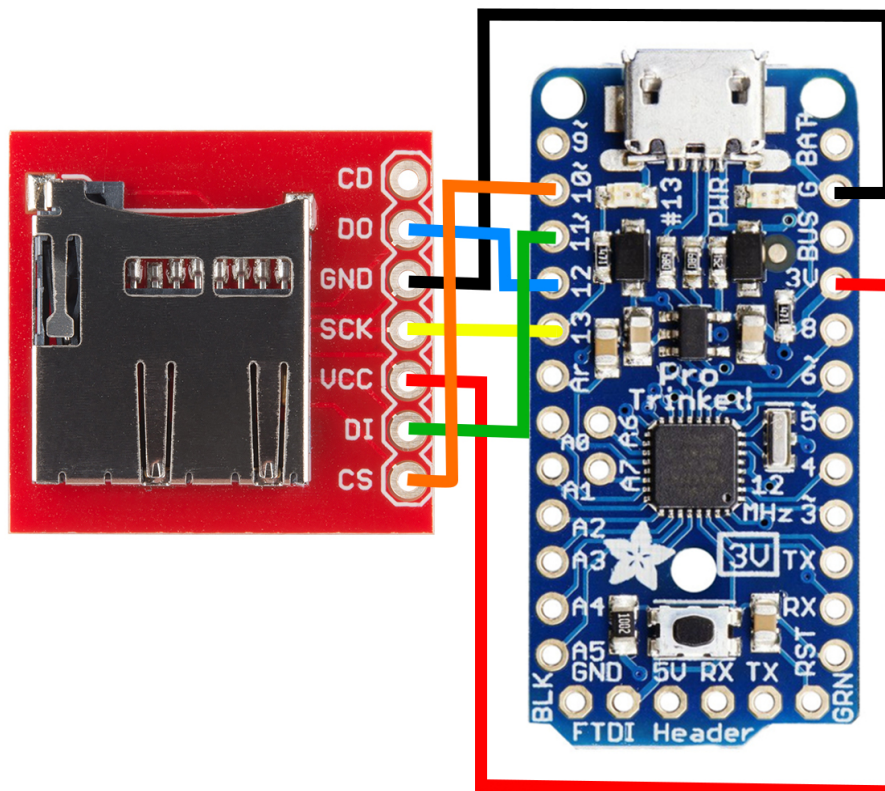
SD Card Integration

In this document, we will connect the SD card breakout to our Arduino by means of a standard SPI connection. We will then insert an SD card, write some text to a file on that SD card, and finally insert the SD card into our computer's card reader so that we can view the text file on our computer.

Part 1 Wiring

The SD Card communicates over an interface called Serial Peripheral Interface or SPI. The SPI bus has three lines for data going from the Arduino to the SD card (green), from the SD card to the Arduino (blue), a shared clock line to ensure synchronous operations (Yellow), and a Chip Select so the Arduino can specify which device on the SPI bus it wants to talk to (Orange). Of course, the device also requires a 3.3V power line (Red) and a ground line (Black).

Wire your SD card to the Arduino as shown in the schematic below. Note that that top pin (CD) on the SD card is not used in this configuration. Also remember that your prototyping boards provide power and ground rails, so you don't need to bring power and ground all the way back to the pins on the Arduino to connect power and ground.



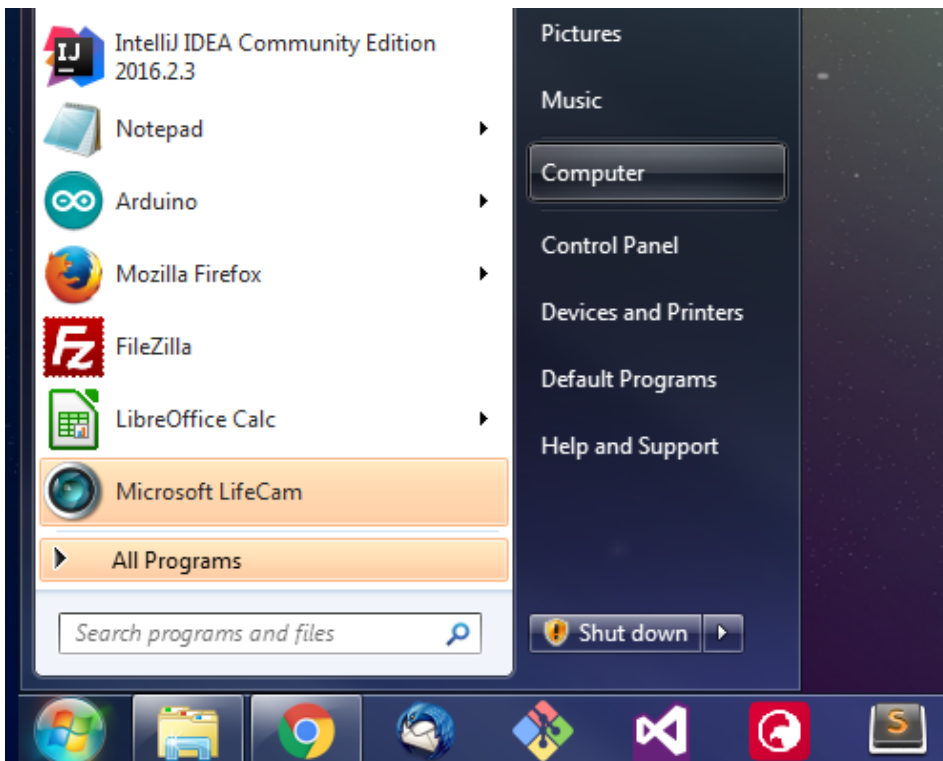
Part 2 SD Card Format

Depending on where you bought your SD card and whether or not it's brand new, you may or may not have it formatted in a way that it can be read from and written to by our code. We will, using our Windows 7 computer, format the disk properly as FAT32.

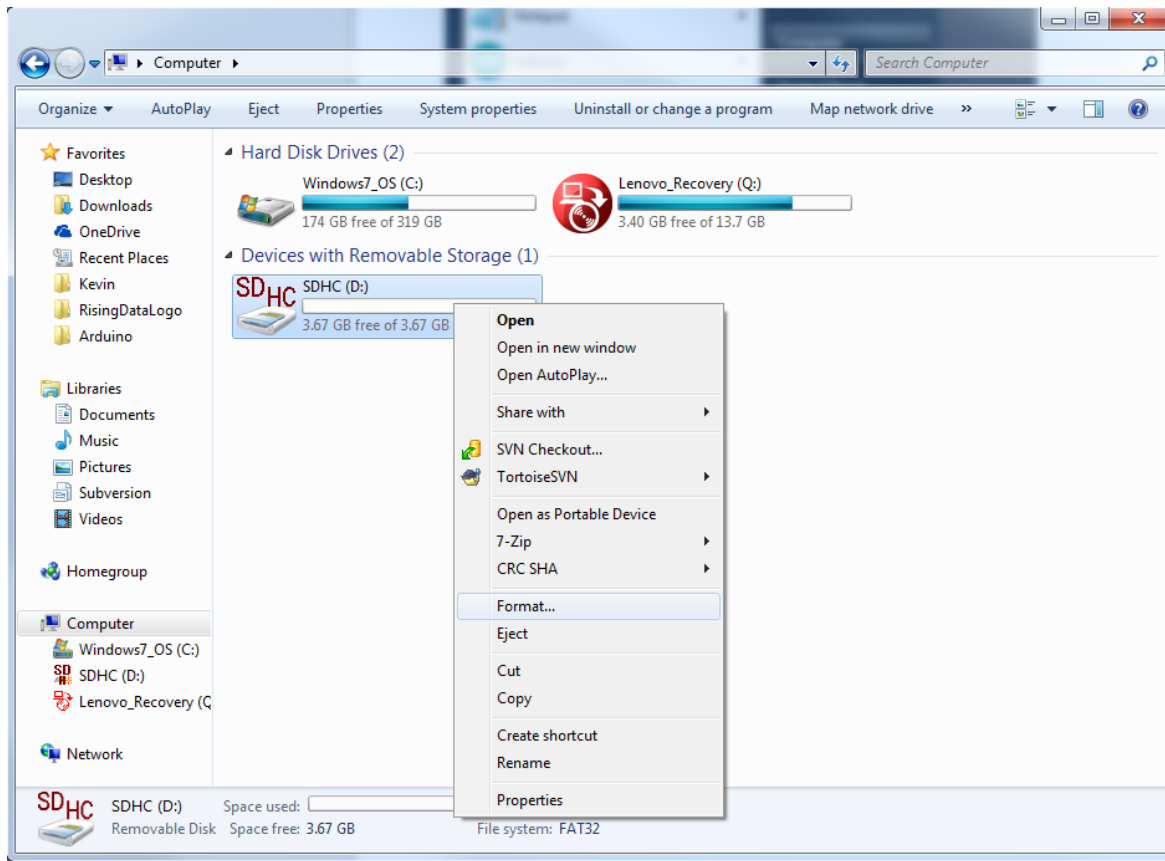
To keep our experiment small, we write to a micro SD card. This is the same type of small form factor storage device you might see in your cell phones or tablets. However, most card readers are not equipped to read a device this small. With your SD card, you will also have received a micro to standard SD adapter as pictures below. This lets your micro SD card fit into your computer's card reader.



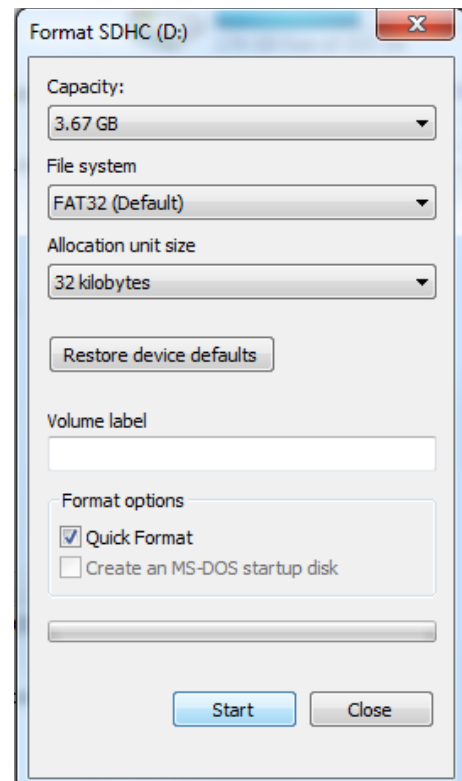
Put the card into your computer's card reader and it should be recognized by windows. Next, go to **Start → Computer**



You should see your device listed under 'Devices and Removable Storage.' Right click on the device, and select 'Format'

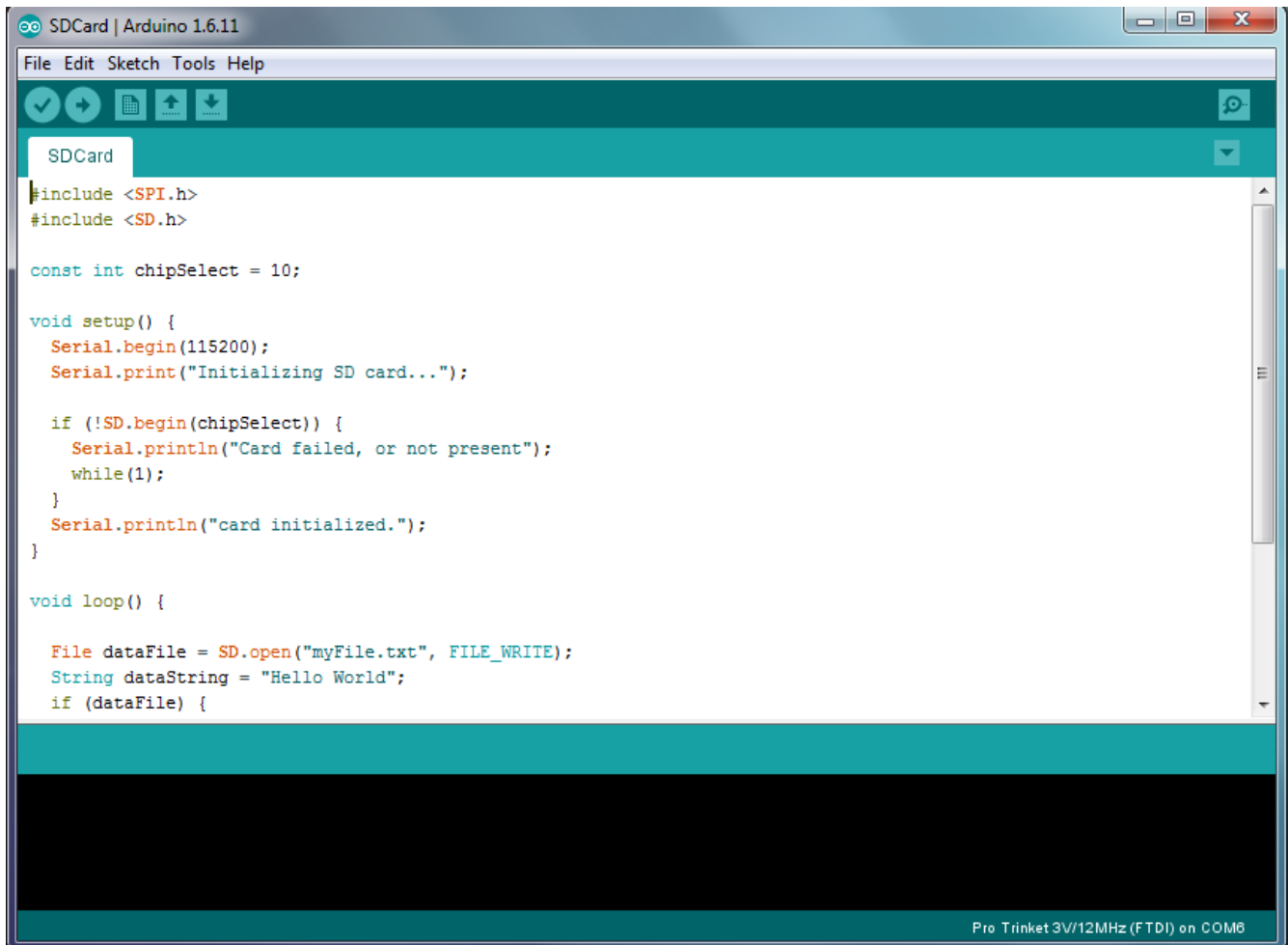


You will be presented with a dialogue window like the one to the right. Make sure the File System shows as FAT32 and then press start. You will be warned (and rightly so) that this process will delete all the information on the drive. If you have any data on this drive, be sure to save it somewhere safe before continuing. When complete, you will get a process complete message. At this point you may remove the SD card from the computer and place it in the SD breakout on your payload.



Part 3 SD Software

Next, open the sketch *RisingData/src/SDCard/SDCard.ino*. You can upload this sketch as written, then open the serial monitor, and see the phrase ‘Hello World’ being written to the screen every second. This text is also getting written into a text file on the SD card.

A screenshot of the Arduino IDE interface. The title bar reads "SDCard | Arduino 1.6.11". The menu bar includes "File", "Edit", "Sketch", "Tools", and "Help". Below the menu bar is a toolbar with icons for checking, uploading, saving, and downloading. The main text area shows the code for "SDCard.ino". The code includes headers for SPI and SD, defines a chipSelect pin, and contains setup and loop functions. The setup function initializes the serial port and the SD card. The loop function opens a file "myFile.txt" and writes "Hello World" to it every second. The status bar at the bottom indicates "Pro Trinket 3V/12MHz (FTDI) on COM6".

```
SDCard | Arduino 1.6.11
File Edit Sketch Tools Help

#include <SPI.h>
#include <SD.h>

const int chipSelect = 10;

void setup() {
  Serial.begin(115200);
  Serial.print("Initializing SD card...");

  if (!SD.begin(chipSelect)) {
    Serial.println("Card failed, or not present");
    while(1);
  }
  Serial.println("card initialized.");
}

void loop() {

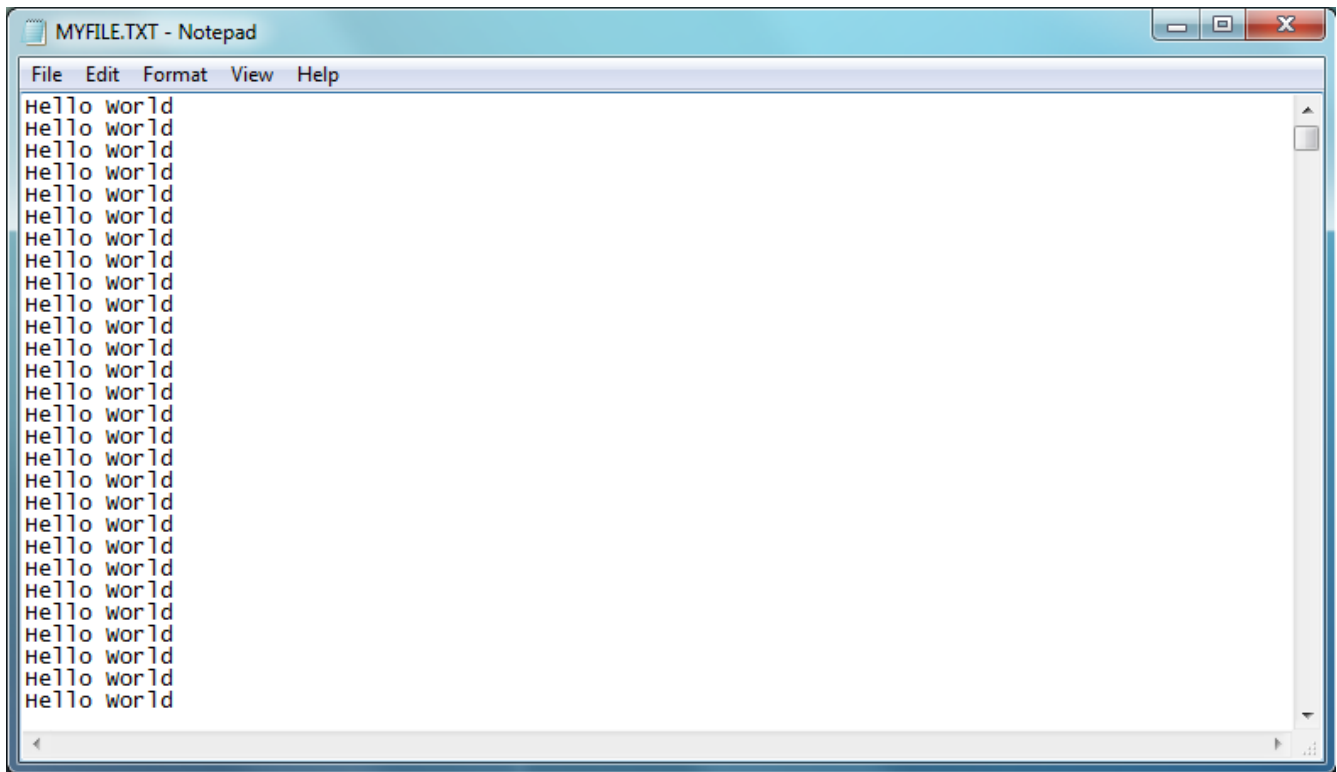
  File dataFile = SD.open("myFile.txt", FILE_WRITE);
  String dataString = "Hello World";
  if (dataFile) {
```

Part 4 Data Analysis

Lastly, we will recover the data from the SD card to our computer so that we can analyze it using the software of our choice. In this example, our ‘data’ is just a simple string being repeated, but in a real-world experiment this would be replaced with measurements from sensors or data from other devices.

- 1) Power down your experiment by disconnecting the USB cable from your computer (or flipping the switch if you are running on batteries).
- 2) Remove the SD card from the socket and place it inside the microSD → SD adapter used in part 2.

3) Place the SD card into your computer's card reader. On the device, you should see a file called MYFILE.TXT with contents similar to this:



Appendix: Code Tour

```
#include <SPI.h>
#include <SD.h>
```

These lines import two libraries. The first, SPI, is the library that implements the SPI interface. The second, SD, is the library that contains the functions which read and write from the SD card using a FAT32 file system.

```
const int chipSelect = 10;
```

The SPI interface requires a dedicated chip select line to be assigned to each and every device on the bus. This line is driven high to put the desired device(s) into a listening state. In our case, we are only using one device so we only have one chip select line. We assigned pin 10, because it happens to be next to the pins for the SPI bus.

```

void setup() {
  Serial.begin(115200);
  Serial.print("Initializing SD card...");

  if (!SD.begin(chipSelect)) {
    Serial.println("Card failed, or not present");
    while(1);
  }
  Serial.println("card initialized.");
}

```

Our setup loop first initializes the serial port for debugging. We use a higher than default baud rate of 115200. This will be useful later in the project. Next we try and initialize the SD card. In our logic, we specify that if the SD card location on the chip select line does not report that it has initialized successfully, we go into an infinite loop to stop program execution. Otherwise, we print that the card initialized and we move on.

```

void loop() {

  File dataFile = SD.open("myFile.txt", FILE_WRITE);
  String dataString = "Hello World";
  if (dataFile) {
    dataFile.println(dataString);
    dataFile.close();
    Serial.println(dataString);
  }
  else {
    Serial.println("error opening file");
  }
  delay(100);
}

```

Our main program loop opens a file on the SD card called 'myFile.txt'. If this file already exists, we will append to the end of that file. If that file does not exist, it will be created for us. Next we create a new string that we define as "Hello World." This text is arbitrary, feel free to experiment with writing whatever words you would like into your file. Next, assuming the file opened successfully, we will write the data into the file and close it. We open and close the file every time we write a piece of data. This ensures data is written in the event our device loses power suddenly. We also print this data to the serial port for debugging. Lastly, we wait 1/10 of a second before repeating everything.