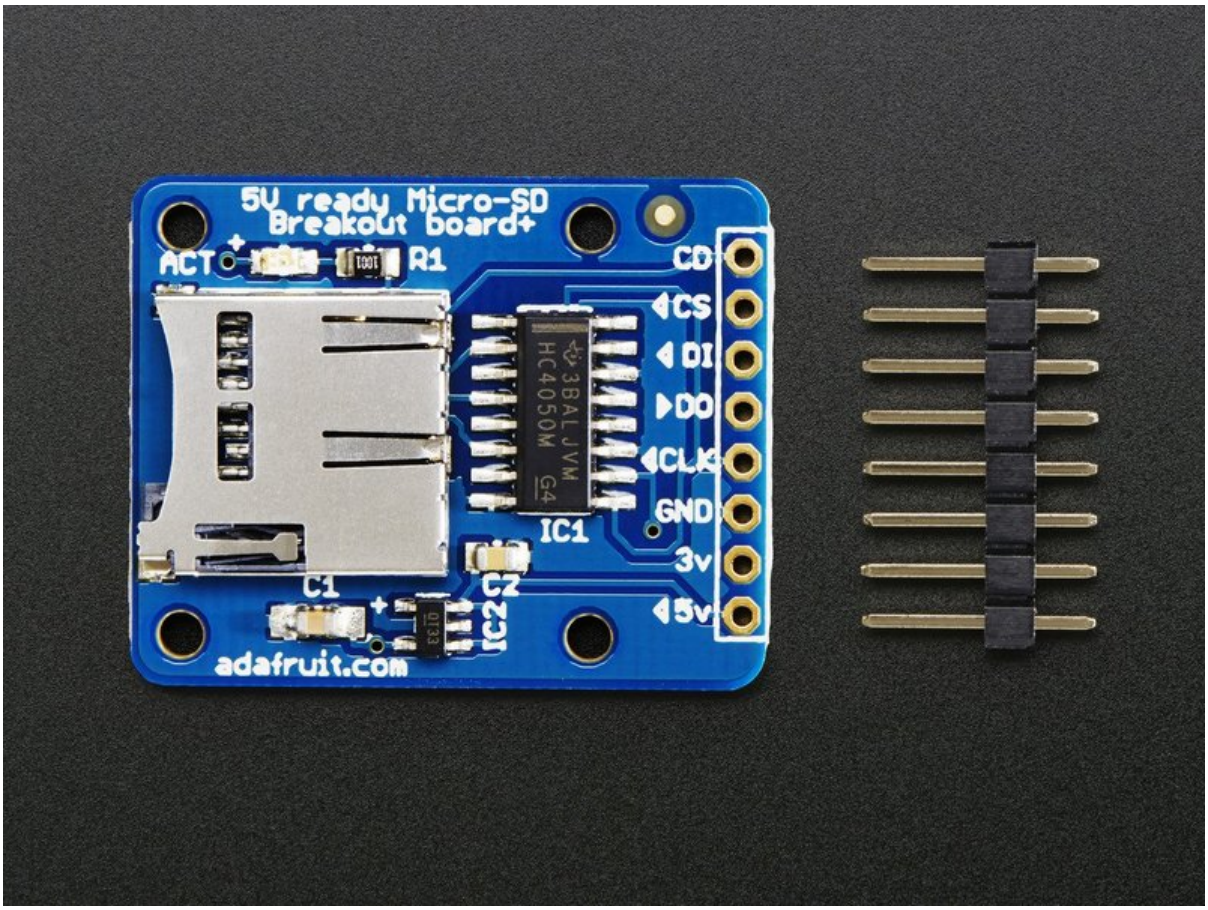
Micro SD Card Breakout Board Tutorial

Created by lady ada



https://learn.adafruit.com/adafruit-micro-sd-breakout-board-card-tutorial

Last updated on 2026-08-12 08:47:01 PM UTC

Table of Contents

Introduction 3

Look out! 5

- What to watch for!

Formatting notes 6

Arduino Wiring 7

Arduino Library 8

- Arduino Library & First Test
- Writing files
- Reading from files
- Recursively listing/reading files

Arduino Library Docs 17

- Other useful functions

Examples 18

- More examples!

CircuitPython 19

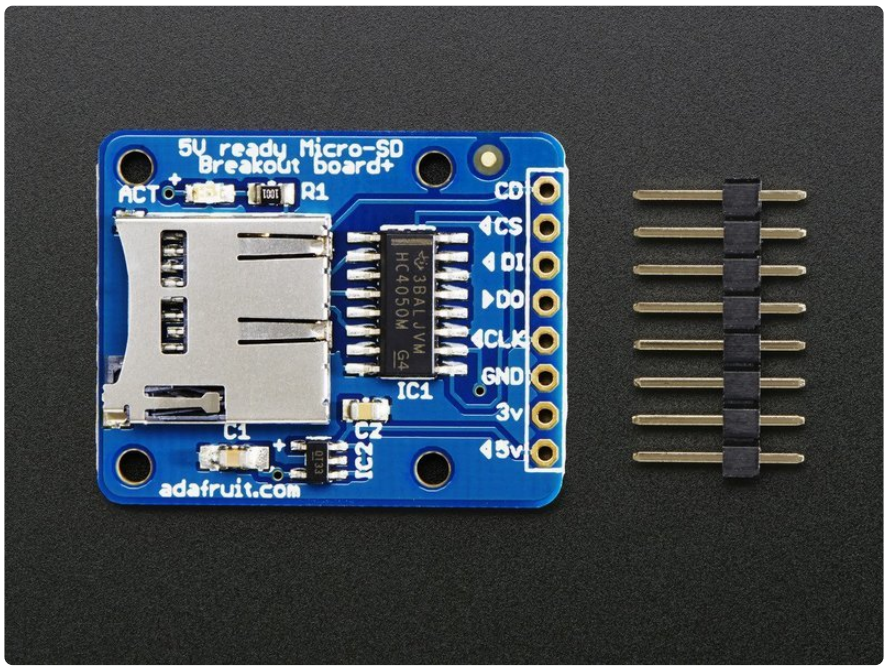
- Adafruit CircuitPython Module Install
- Usage
- Initialize & Mount SD Card Filesystem Using sdcardio
- Initialize & Mount SD Card Filesystem Using adafruit_sdcard
- Reading & Writing Data
- List Files
- Logging Temperature

Python/Linux 29

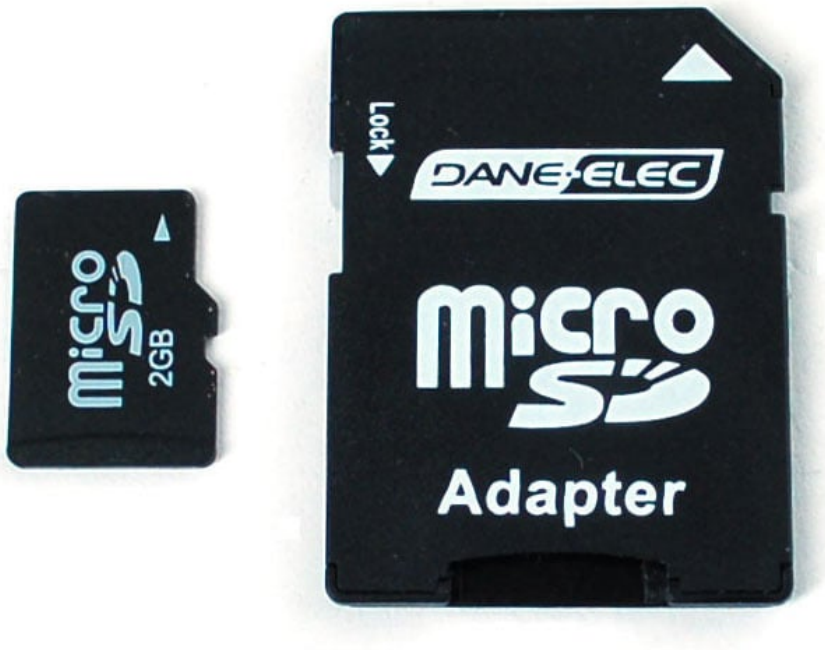
Download 30

- Schematic
- Fabrication Print

Introduction



If you have a project with any audio, video, graphics, data logging, etc in it, you'll find that having a removable storage option is essential. Most microcontrollers have extremely limited built-in storage. For example, even the Arduino Mega chip (the Atmega2560) has a mere 4Kbytes of EEPROM storage. There's more flash (256K) but you can't write to it as easily and you have to be careful if you want to store information in flash that you don't overwrite the program itself!



If you're doing any sort of data logging, graphics or audio, you'll need at least a megabyte of storage, and 64 M is probably the minimum. To get that kind of storage we're going to use the same type that's in every digital camera and mp3 player: flash cards! Often called SD or microSD cards, they can pack **gigabytes** into a space smaller than a coin. They're also available in every electronics shop so you can easily get more and best of all, many computers have SD or microSD card readers built in so you can move data back and forth between say your Arduino GPS data logger and your computer graphing software:

Look out!

What to watch for!

There are a few things to watch for when interacting with SD cards:

One is that they are strictly 3.3V devices and the power draw when writing to the card can be fairly high, up to 100mA (or more)! That means that you **must** have a fairly good 3.3V power supply for the card. Secondly you must also have 3.3V logic to interface to the pins. We've found that SD cards are fairly sensitive about the interface pins - the newest cards are edge triggered and require very 'square' transitions - things like resistor dividers and long wires will have a deleterious effect on the transition speed, so **keep wires short, and avoid using resistor dividers for the 3.3V logic lines**. We suggest instead using level shifters, such as **HEF4050**, **74LVX245** or **74AHC125** chips.

For the level shifter we use the CD74HC4050 (https://adafru.it/Boj) which has a typical propagation delay of ~10ns

Secondly, there are two ways to interface with SD cards - **SPI mode** and **SDIO mode**. SDIO mode is faster, but is more complex and as far as we can tell, requires signing non-disclosure documents. For that reason, you will likely never encounter SDIO mode interface code. Instead, every SD card has a 'lower speed' SPI mode that is

easy for any microcontroller to use. SPI mode requires four pins (we'll discuss them in detail later) so it's not pin-heavy like some parallel-interface components

SD cards come in two popular flavors - **microSD** and **SD**. The interface, code, structure, etc is all the same. The only differences is the size. MicroSD are much much smaller in physical size.

Third, SD cards are 'raw' storage. They're just sectors in a flash chip, there's no structure that you have to use. That means you could format an SD card to be a Linux filesystem, a FAT (DOS) filesystem or a Mac filesystem. You could also not have any filesystem at all! However, 99% of computers, cameras, MP3 players, GPS loggers, etc require **FAT16** or **FAT32** for the filesystem. The tradeoff here is that for smaller microcontrollers (like the Arduino) the addition of the complex file format handling can take a lot of flash storage and RAM.

Formatting notes

Even though you can/could use your SD card 'raw' - it's most convenient to format the card to a filesystem. For the Arduino library we'll be discussing, and nearly every other SD library, the card must be formatted FAT16 or FAT32. Some only allow one or the other. The Arduino SD library can use either.

If you bought an SD card, chances are it's already pre-formatted with a FAT filesystem. However you may have problems with how the factory formats the card, or if it's an old card it needs to be reformatted. The Arduino SD library we use supports both **FAT16** and **FAT32** filesystems. If you have a very small SD card, say 8-32 Megabytes you might find it is formatted **FAT12** which isn't supported. You'll have to reformat these cards. Either way, it's **always** good idea to format the card before using, even if it's new! Note that formatting will erase the card so save anything you want first.

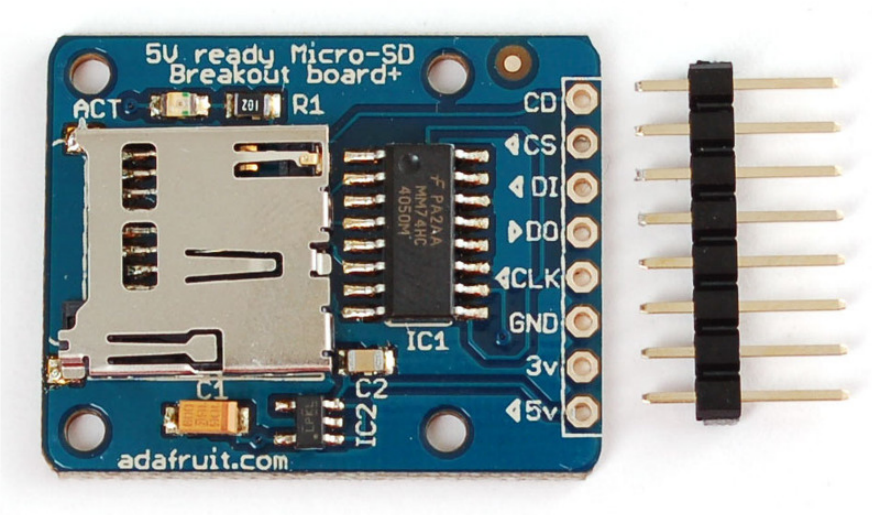
Text emphasized with a red exclamation:	We strongly recommend you use the official SD card formatter utility - written by the SD association it solves many problems that come with bad formatting!

[Download the formatter from https://www.sdcard.org/downloads/formatter/](https://adafru.it/FKd) (https://adafru.it/FKd)

Download it and run it on your computer, there's also a manual linked from that page for use.

Arduino Wiring

Now that your card is ready to use, we can wire up the microSD breakout board! The breakout board we designed takes care of a lot for you. There's an onboard ultra-low dropout regulator that will convert voltages from 3.3V-6v down to ~3.3V (**IC2**). There's also a level shifter that will convert the interface logic from 3.3V-5V to 3.3V. That means you can use this board to interact with a 3.3V or 5V microcontrollers.



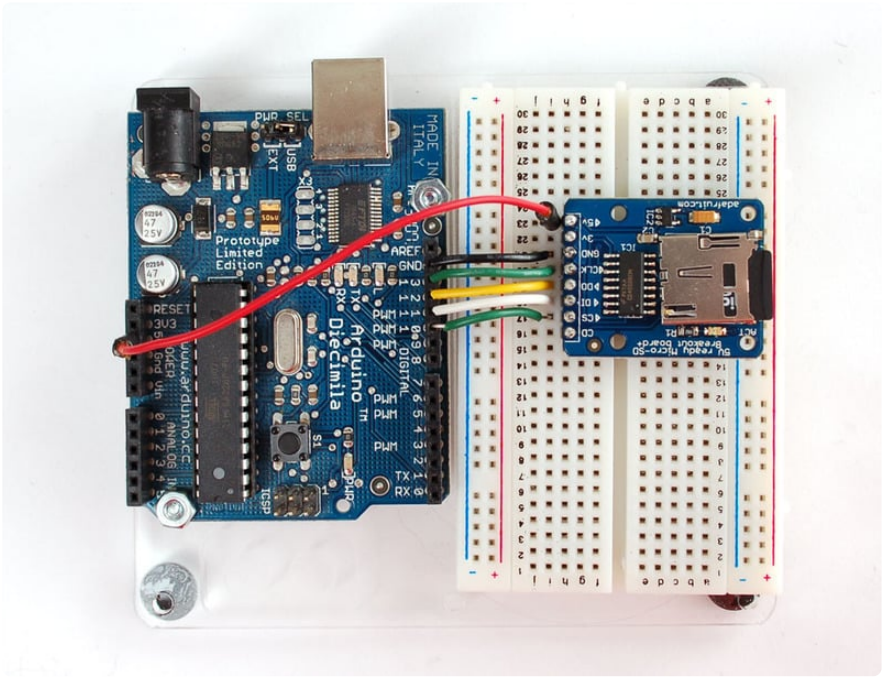
In this tutorial we will be using an Arduino to demonstrate the wiring and interfacing. If you have another microcontroller, you'll need to adapt the wiring and code to match!

Because SD cards require a lot of data transfer, they will give the best performance when connected up to the **hardware** SPI pins on a microcontroller. The hardware SPI pins are much faster than 'bit-banging' the interface code using another set of pins. For 'classic' Arduinos such as the Duemilanove/Diecimila/Uno those pins are **digital 13 (SCK), 12 (MISO)** and **11 (MOSI)**. You will also need a fourth pin for the 'chip/secondary select' (**SS**) line. Traditionally this is pin **10** but you can actually use any pin you like. If you have a Mega, the pins are different! You'll want to use digital **50 (MISO), 51 (MOSI), 52 (SCK),** and for the CS line, the most common pin is **53 (SS).** Again, you can change the SS (pin **10** or **53**) later but for now, stick with those pins.

- Connect the **5V** pin to the **5V** pin on the Arduino
- Connect the **GND** pin to the **GND** pin on the Arduino

- Connect **CLK** to pin **13** or **52**
- Connect **DO** to pin **12** or **50**
- Connect **DI** to pin **11** or **51**
- Connect **CS** to pin **10** or **53**

There's one more pin **CD** - this is the Card Detect pin. It shorts to ground when a card is not inserted. (Note that some card holders are the other way around). You should connect a pull up resistor (10K or so) and wire this to another pin if you want to detect when a card is inserted. We won't be using it for now.



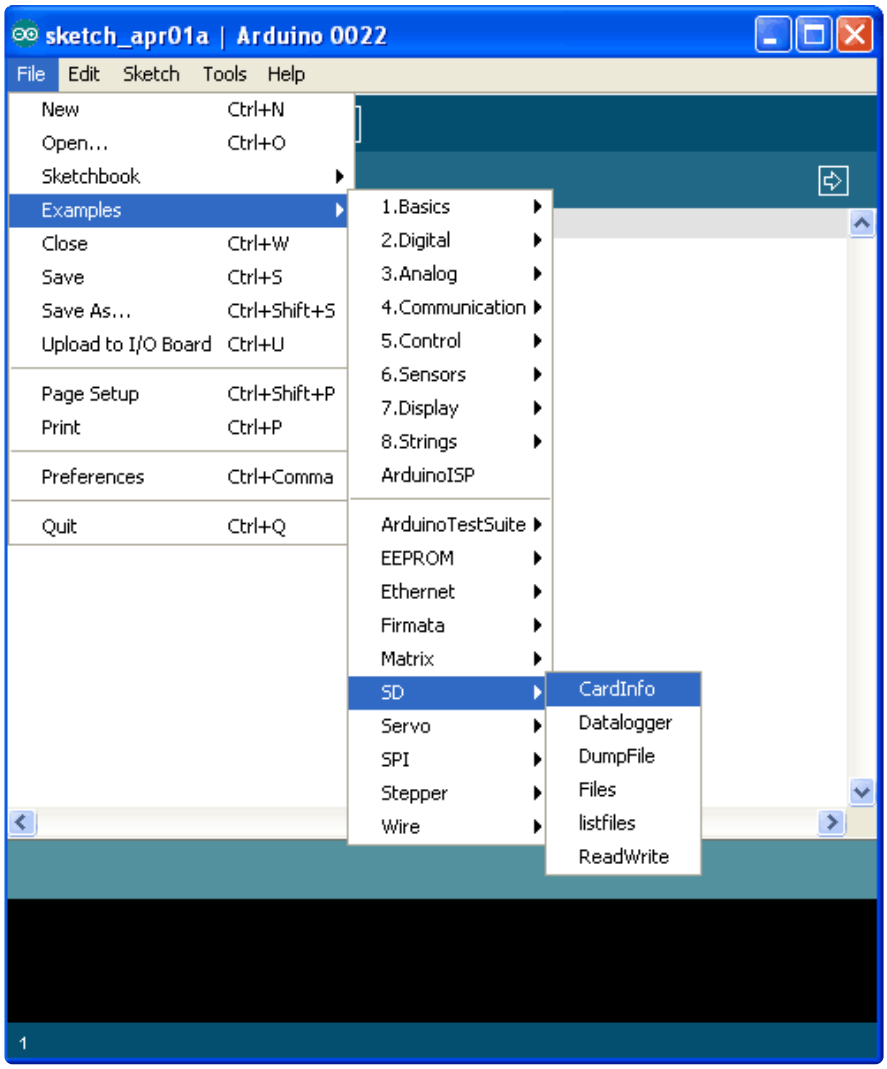
That's it! Now you're ready to rock!

Arduino Library

Arduino Library & First Test

Interfacing with an SD card is a bunch of work, but luckily for us, Adafruit customer fat16lib (William G) has written a very nice Arduino library just for this purpose and it's now part of the Arduino IDE known as **SD** (pretty good name, right?) You can see it in the Examples submenu

Next, select the **CardInfo** example sketch.

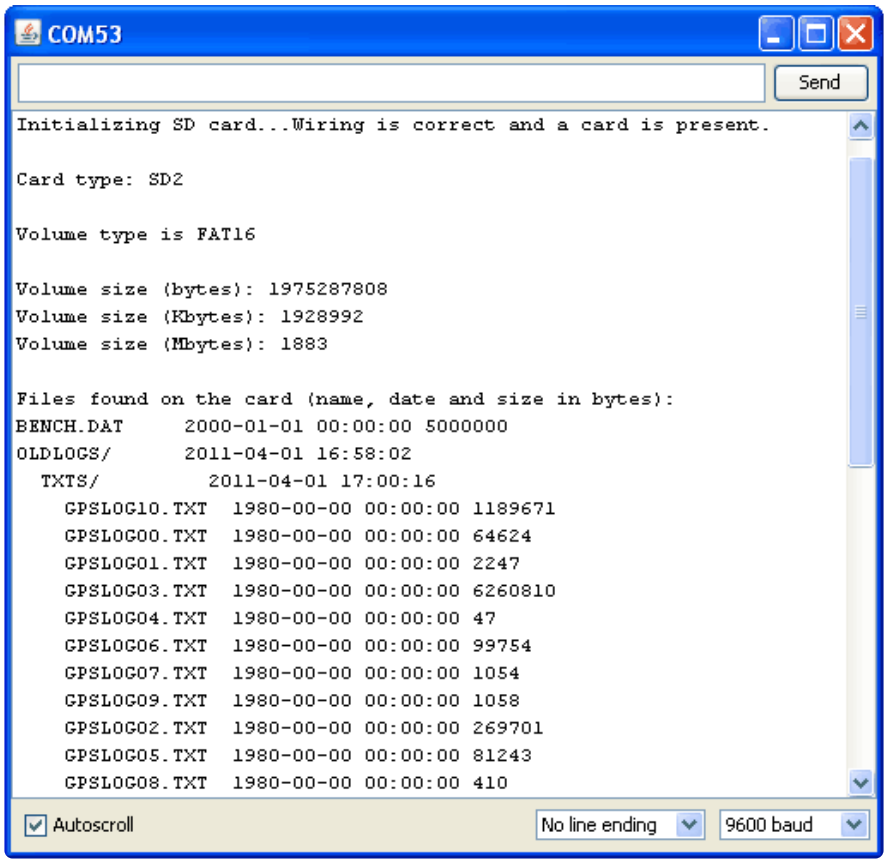


This sketch will not write any data to the card, just tell you if it managed to recognize it, and some information about it. This can be **very** useful when trying to figure out whether an SD card is supported. Before trying out a new card, please try out this sketch!

Go to the beginning of the sketch and make sure that the **chipSelect** line is correct, for this wiring we're using digital pin 10 so change it to 10!

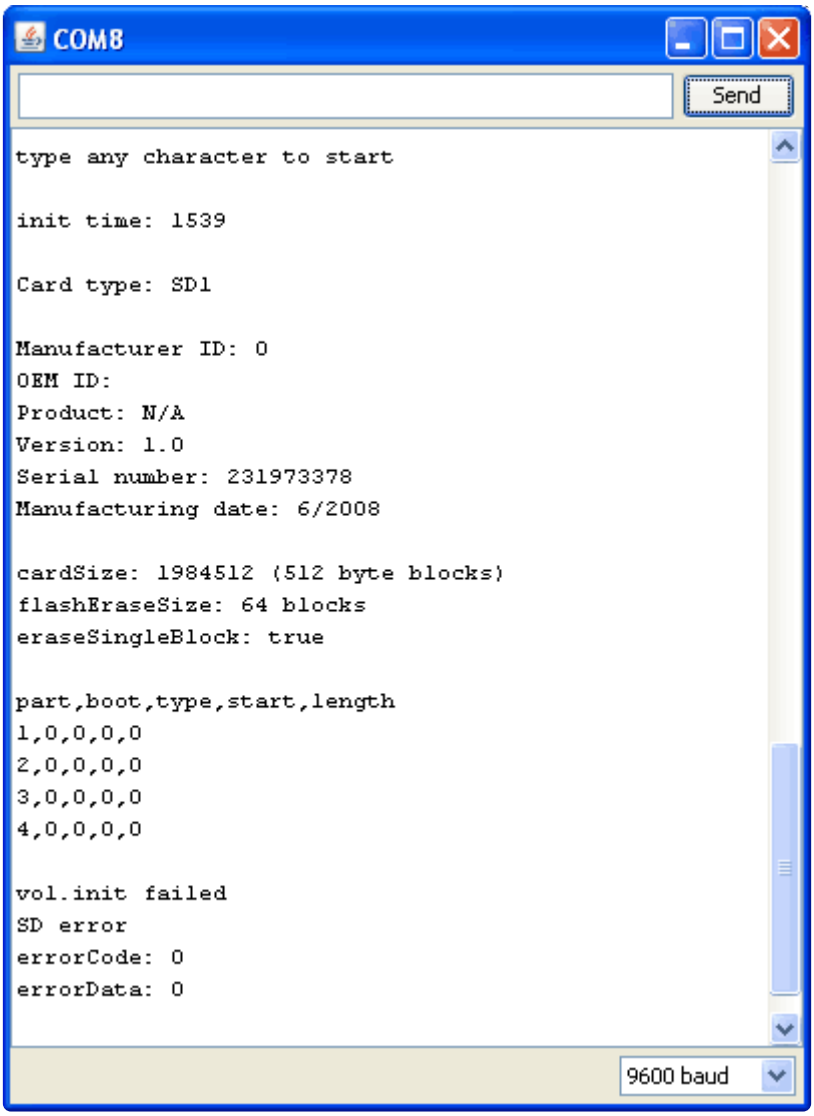
OK, now insert the SD card into the breakout board and upload the sketch.

Open up the Serial Monitor and type in a character into the text box (& hit send) when prompted. You'll probably get something like the following:



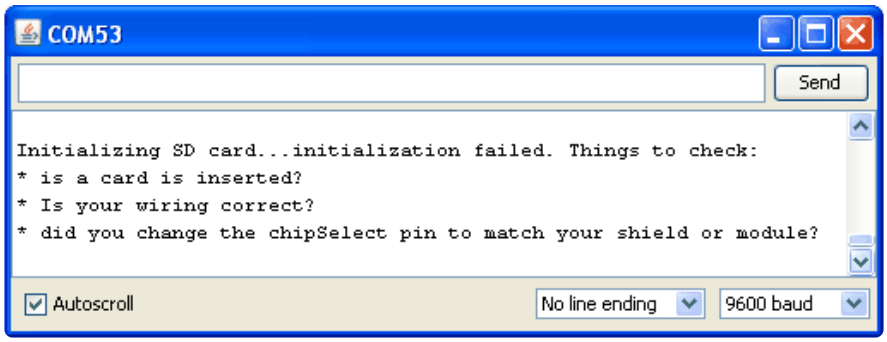
It's mostly gibberish, but it's useful to see the **Volume type is FAT16** part as well as the size of the card (about 2 GB which is what it should be) etc.

If you have a bad card, which seems to happen more with ripoff version of good brands, you might see:



The card mostly responded, but the data is all bad. Note that the **Product ID** is **"N/A"** and there is no **Manufacturer ID** or **OEM ID**. This card returned some SD errors. It's basically a bad scene, I only keep this card around to use as an example of a bad card! If you get something like this (where there is a response but it's corrupted) you can try to reformat it or if it still flakes out, should toss the card.

Finally, try taking out the SD card and running the sketch again, you'll get the following,



It couldn't even initialize the SD card. This can also happen if there's a soldering or wiring error or if the card is really damaged.

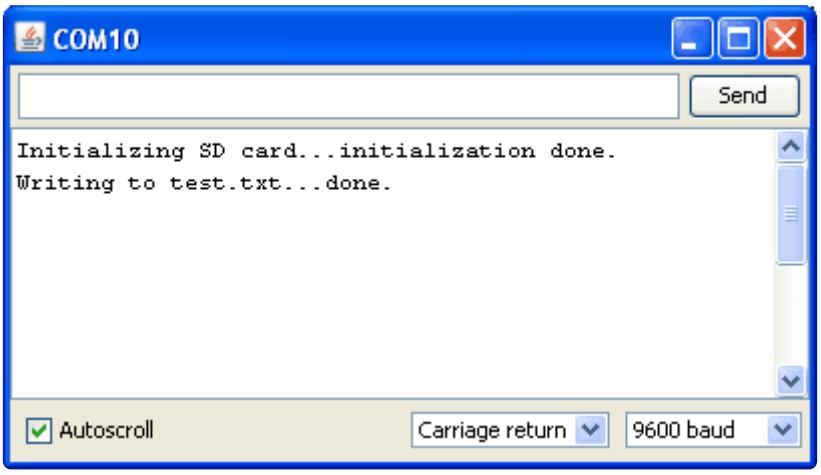
Writing files

The following sketch will do a basic demonstration of writing to a file. This is a common desire for datalogging and such.

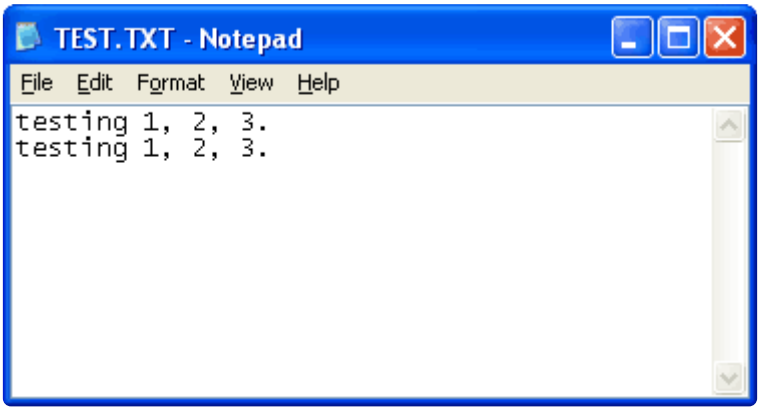
```
#include <SD.h>

File myFile;

void setup()
{
  Serial.begin(9600);
  Serial.print("Initializing SD card...");
  // On the Ethernet Shield, CS is pin 4. It's set as an output by default.
  // Note that even if it's not used as the CS pin, the hardware SS pin
  // (10 on most Arduino boards, 53 on the Mega) must be left as an output
  // or the SD library functions will not work.
   pinMode(10, OUTPUT);

  if (!SD.begin(10)) {
    Serial.println("initialization failed!");
    return;
  }
  Serial.println("initialization done.");

  // open the file. note that only one file can be open at a time,
  // so you have to close this one before opening another.
  myFile = SD.open("test.txt", FILE_WRITE);

  // if the file opened okay, write to it:
  if (myFile) {
    Serial.print("Writing to test.txt...");
    myFile.println("testing 1, 2, 3.");
    // close the file:
    myFile.close();
    Serial.println("done.");
  } else {
    // if the file didn't open, print an error:
```

```
34      Serial.println("error opening test.txt");
35    }
36  }
37
38  void loop()
39  {
40      // nothing happens after setup
41  }
```

When you run it you should see the following:



You can then open up the file in your operating system by inserting the card. You'll see one line for each time the sketch ran. That is to say, it **appends** to the file, not overwriting it.



Some things to note:

- You can have multiple files open at a time, and write to each one as you wish.

- You can use **print** and **println()** just like Serial objects, to write strings, variables, etc
- You must **close()** the file(s) when you're done to make sure all the data is written permanently!
- You can open files in a directory. For example, if you want to open a file in the directory such as **/MyFiles/example.txt** you can call **SD.open("/myfiles/example.txt")** and it will do the right thing.

Text emphasized with a red exclamation:

The SD card library does not support 'long filenames' such as we are used to. Instead, it uses the 8.3 format for file names, so keep file names short! For example IMAGE.JPG is fine, and datalog.txt is fine but "My GPS log file.text" is not! Also keep in mind that short file names do not have 'case' sensitivity, so datalog.txt is the same file as DataLog.Txt is the same file as DATALOG.TXT

Reading from files

Next up we will show how to read from a file, it's very similar to writing in that we **SD.open()** the file but this time we don't pass in the argument **FILE_WRITE** this will keep you from accidentally writing to it. You can then call **available()** (which will let you know if there is data left to be read) and **read()** from the file, which will return the next byte.

```
#include <SD.h>

File myFile;

void setup()
{
  Serial.begin(9600);
  Serial.print("Initializing SD card...");
  // On the Ethernet Shield, CS is pin 4. It's set as an output by default.
  // Note that even if it's not used as the CS pin, the hardware SS pin
  // (10 on most Arduino boards, 53 on the Mega) must be left as an output
  // or the SD library functions will not work.
   pinMode(10, OUTPUT);

  if (!SD.begin(10)) {
    Serial.println("initialization failed!");
    return;
  }
  Serial.println("initialization done.");

```

```
21    // open the file for reading:
22    myFile = SD.open("test.txt");
23    if (myFile) {
24      Serial.println("test.txt:");
25
26      // read from the file until there's nothing else in it:
27      while (myFile.available()) {
28          Serial.write(myFile.read());
29      }
30      // close the file:
31      myFile.close();
32    } else {
33      // if the file didn't open, print an error:
34      Serial.println("error opening test.txt");
35    }
36  }
37
38  void loop()
39  {
40      // nothing happens after setup
41  }
```

Some things to note:

- You can have multiple files open at a time, and read from each one as you wish.
- **Read()** only returns a byte at a time. It does not read a full line or a number!
- You should **close()** the file(s) when you're done to reduce the amount of RAM used.

Text emphasized with a red exclamation:

The SD card library does not support 'long filenames' such as we are used to. Instead, it uses the 8.3 format for file names, so keep file names short! For example IMAGE.JPG is fine, and datalog.txt is fine by "My GPS log file.text" is not! Also keep in mind that short file names do not have 'case' sensitivity, so datalog.txt is the same file as DataLog.Txt is the same file as DATALOG.TXT

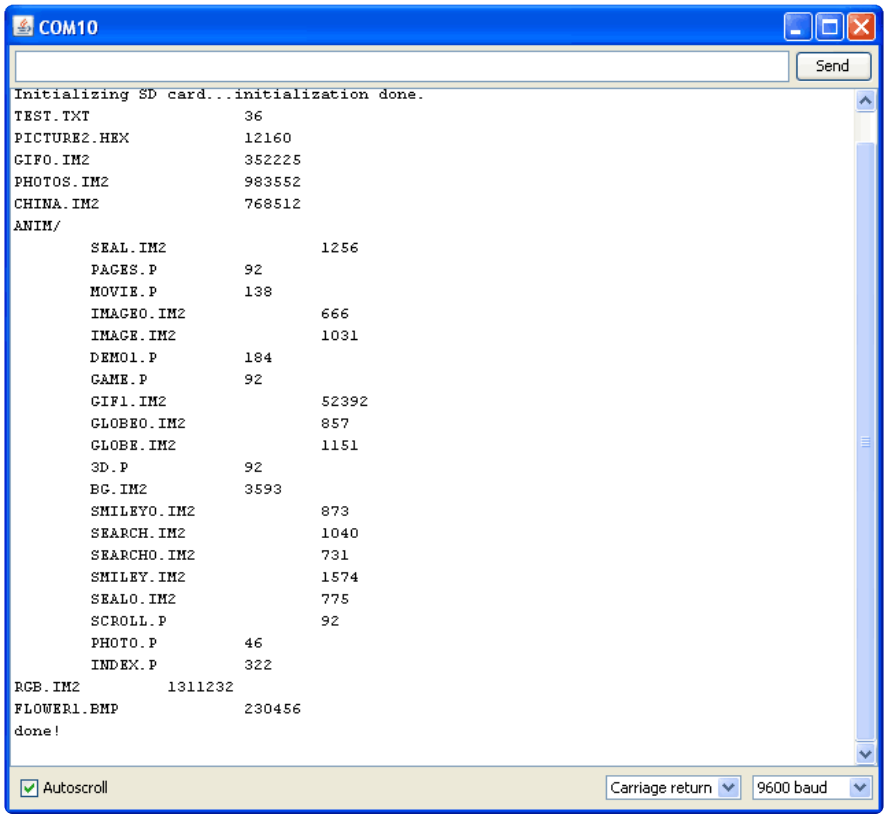
Recursively listing/reading files

The last example we have shows more advanced use. A common request is for example wanting to list every file on the SD card, or play ever music file or similar. In the latest version of the SD library, you can recurse through a directory and call

openNextFile() to get the next available file. These aren't in alphabetical order, they're in order of creation so just watch out for that!

To see it, run the **SD→listfiles** example sketch

Here you can see that we have a subdirectory **ANIM** (we have animation files in it). The numbers after each file name are the size in bytes of the file. This sketch is handy if you want to check what files are called on your card. The sketch also demonstrates how to do directory handling.



Arduino Library Docs

Other useful functions

There's a few useful things you can do with **SD** objects we'll list a few here:

- If you just want to check if a file exists, use **SD.exists("filename.txt")** which will return true or false.

- You can delete a file by calling **SD.remove("unwanted.txt")** - be careful! This will really delete it, and there's no 'trash can' to pull it out of.
- You can create a subdirectory by calling **SD.mkdir("/mynewdir")** handy when you want to stuff files in a location. Nothing happens if it already exists but you can always call **SD.exists()** above first.

Also, there's a few useful things you can do with **File** objects:

- You can **seek()** on a file. This will move the reading/writing pointer to a new location. For example **seek(0)** will take you to the beginning of the file, which can be very handy!
- Likewise you can call **position()** which will tell you where you are in the file.
- If you want to know the size of a file, call **size()** to get the number of bytes in the file.
- Directories/folders are special files, you can determine if a file is a directory by calling **isDirectory()**
- Once you have a directory, you can start going through all the files in the directory by calling **openNextFile()**
- You may end up with needing to know the name of a file, say if you called **openNextFile()** on a directory. In this case, call **name()** which will return a pointer to the 8.3-formatted character array you can directly **Serial.print()** if you want.

Examples

More examples!

If you want to use an SD card for datalogging, we suggest checking out our Datalogging shield (https://adafru.it/dpH) and GPS logging shield (https://adafru.it/dpI) - there's example code specifically for those purposes.

If you want to use the SD card for loading images (such as for a color display) look at our 2.8" TFT shield (https://adafru.it/dpJ) and 1.8" TFT breakout tutorials (https://adafru.it/ckK). Those have examples of how we read BMP files off disk and parse them.

CircuitPython

Text emphasized with a yellow exclamation:

As of CircuitPython 9, a mount point (folder) named **/sd** is required on the **CIRCUITPY** drive. Make sure to create that directory after upgrading CircuitPython.

https://adafru.it/19ei

Adafruit CircuitPython Module Install

To use a microSD card with your Adafruit CircuitPython board you'll need to install the Adafruit_CircuitPython_SD (https://adafru.it/zwC) module on your board.

First make sure you are running the latest version of Adafruit CircuitPython (https://adafru.it/Amd) for your board.

Next you'll need to install the necessary libraries to use the hardware--carefully follow the steps to find and install these libraries from Adafruit's CircuitPython library bundle (https://adafru.it/zdx). Our introduction guide has a great page on how to install the library bundle (https://adafru.it/ABU) for both express and non-express boards. **Be sure to use the latest CircuitPython Bundle** as it includes an updated version of the SD card module with a few necessary fixes!

If your board supports **sdcardio** (https://adafru.it/-Cy), then this is the preferred method to do things. **sdcardio** is a built-in module on boards that support it, so you don't have to copy it over.

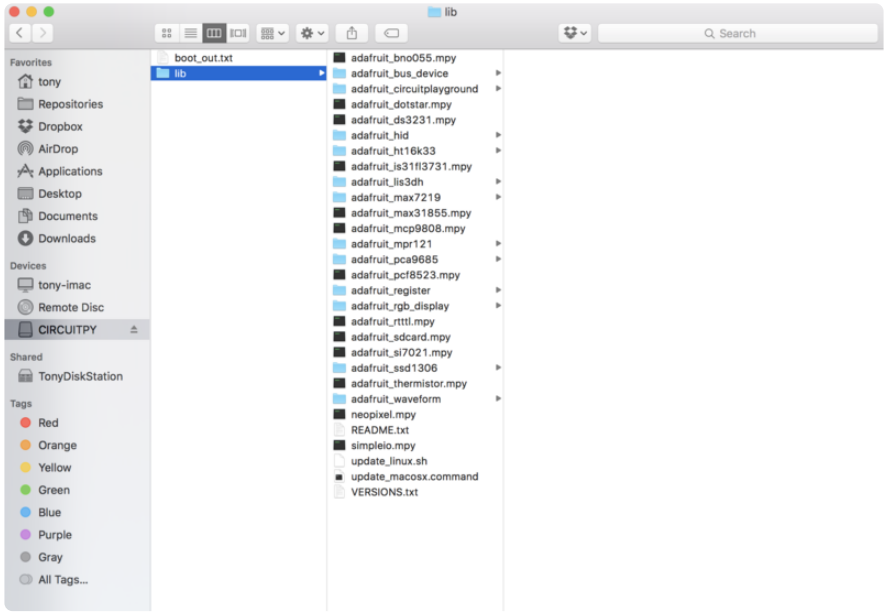
Remember for non-express boards like the Feather M0 Basic, you'll need to manually install the necessary libraries from the bundle:

- **adafruit_sdcard.mpy**
- **adafruit_bus_device**

If your board doesn't support USB mass storage, like the ESP8266, then use a tool like ampy to copy the file to the board (https://adafru.it/s1f). You can use the latest

version of ampy and its new directory copy command (https://adafru.it/q2A) to easily move module directories to the board.

Before continuing make sure your board's lib folder or root filesystem has the **adafruit_sdcard.mpy** and **adafruit_bus_device** modules copied over.



Usage

The following section will show how to initialize the SD card and read & write data to it from the board's Python prompt / REPL.

Next connect to the board's serial REPL (https://adafru.it/Awz)so you are at the CircuitPython >>> prompt.

Text emphasized with a blue exclamation:

sdcardio is an SPI interface SD card library in CircuitPython 6.0 that is optimized in C to be much faster than the original adafruit_sdcard library. Some boards don't have enough memory for this module, in which case you will have to use adafruit_sdcard

Initialize & Mount SD Card Filesystem Using **sdcardio**

Before using this method, verify that your board supports sdcardio using the support matrix (https://adafru.it/-Cy). If it does not, try using **adafruit_sdcard** instead.

Before you can use the microSD card you need to initialize its SPI connection and mount its filesystem. First import all the modules we'll need:

```
import board
import busio
import sdcardio
import storage
```

Next create the SPI bus and a digital output for the microSD card's chip select line (be sure to select the right pin names for your wiring):

```
# Use the board's primary SPI bus
spi = board.SPI()
# Or, use an SPI bus on specific pins:
#spi = busio.SPI(board.SD_SCK, MOSI=board.SD_MOSI, MISO=board.SD_MISO)

# For breakout boards, you can choose any GPIO pin that's convenient:
cs = board.D10
# Boards with built in SPI SD card slots will generally have a
# pin called SD_CS:
#cs = board.SD_CS
```

Text emphasized with a yellow exclamation:

If you are using some non-standard CircuitPython board, you may need to use specific GPIO pins for MISO and MOSI to get the SPI recognized appropriately

Text emphasized with a blue exclamation:

If your SPI chip select pin is different or not defined like SD_CS, put the appropriate pin number in the code, like board.D5.

Note that when you use `sdcardio`, `cs` is a `Pin` object, not a `DigitalInOut` object. If you change your code to use `adafruit_sdcard`, you need to use a `DigitalInOut` object instead.

At this point you're ready to create the microSD card object and the filesystem object:

```
sdcard = sdcardio.SDCard(spi, cs)
vfs = storage.VfsFat(sdcard)
```

Notice the `sdcardio` module has a `SDCard` class which contains all the logic for talking to the microSD card at a low level. This class needs to be told the SPI bus and chip select pin in its constructor.

After a `SDCard` instance is created it can be passed to the `storage` module's `VfsFat` class. This class has all the logic for translating CircuitPython filesystem calls into low level microSD card access. Both the `SDCard` and `VfsFat` class instances are required to mount the card as a new filesystem.

Finally you can mount the microSD card's filesystem into the CircuitPython filesystem. For example to make the path **/sd** on the CircuitPython filesystem read and write from the card run this command:

```
storage.mount(vfs, "/sd")
```

At this point, you can read and write to the SD card using common Python functions like `open`, `read`, and `write`. The filenames will all begin with `"/sd/"` to differentiate them from the files on the **CIRCUITPY** drive. If you're not familiar, and all this worked, skip ahead to the section labeled **Reading & Writing Data**.

Text emphasized with a green exclamation:

If the same SPI bus is shared with other peripherals, it is important that the SD card be initialized before accessing any other peripheral on the bus. Failure to do so can prevent the SD card from being recognized until it is powered off or re-inserted.

Initialize & Mount SD Card Filesystem Using **adafruit_sdcard**

Before you can use the microSD card you need to initialize its SPI connection and mount its filesystem. First import the necessary modules to initialize the SPI and CS line physical connections:

```
import board
import busio
import digitalio
```

Next create the SPI bus and a digital output for the microSD card's chip select line (be sure to select the right pin name or number for your wiring):

```
spi = busio.SPI(board.SCK, MOSI=board.MOSI, MISO=board.MISO)
# Use board.SD_CS for Feather M0 Adalogger
cs = digitalio.DigitalInOut(board.SD_CS)
# Or use a digitalio pin like 5 for breakout wiring:
#cs = digitalio.DigitalInOut(board.D5)
```

Now import modules to access the SD card and filesystem:

```
import adafruit_sdcard
import storage
```

At this point you're ready to create the microSD card object and the filesystem object:

```
sdcard = adafruit_sdcard.SDCard(spi, cs)
vfs = storage.VfsFat(sdcard)
```

Notice the **adafruit_sdcard** module has a **SDCard** class which contains all the logic for talking to the microSD card at a low level. This class needs to be told the SPI bus and chip select digital IO pin in its initializer.

After a **SDCard** class is created it can be passed to the **storage** module's **VfsFat** class. This class has all the logic for translating CircuitPython filesystem calls into low level microSD card access. Both the **SDCard** and **VfsFat** class instances are required to mount the card as a new filesystem.

Finally you can mount the microSD card's filesystem into the CircuitPython filesystem. For example to make the path **/sd** on the CircuitPython filesystem read and write from the card run this command:

```
storage.mount(vfs, "/sd")
```

The first parameter to the storage.mount command is the **VfsFat** class instance that was created above, and the second parameter is the location within the CircuitPython filesystem that you'd like to 'place' the microSD card. Remember the mount location as you'll need it to read and write files on the card!

Text emphasized with a green exclamation:

If the same SPI bus is shared with other peripherals, it is important that the SD card be initialized before accessing any other peripheral on the bus. Failure to do so can prevent the SD card from being recognized until it is powered off or re-inserted.

Reading & Writing Data

Once the microSD card is mounted inside CircuitPython's filesystem you're ready to read and write data from it. Reading and writing data is simple using Python's file operations like open (https://adafru.it/reL), close (https://adafru.it/ryE), read (https://adafru.it/ryE), and write (https://adafru.it/ryE). The beauty of CircuitPython and MicroPython is that they try to be as similar to desktop Python as possible, including access to files.

For example to create a file and write a line of text to it you can run:

```
with open("/sd/test.txt", "w") as f:
    f.write("Hello world!\r\n")
```

Notice the with statement is used to create a context manager that opens and automatically closes the file. This is handy because with file access you Python you must close the file when you're done or else all the data you thought was written might be lost!

The open function is used to open the file by telling it the path to it, and the mode (w for writing). Notice the path is under **/sd**, **/sd/test.txt**. This means the file will be created on the microSD card that was mounted as that path.

Text emphasized with a yellow exclamation:

Starting with CircuitPython 9, the SD card MUST be mapped to /sd on the CircuitPython file system. Create an empty sd directory in the root directory of the CIRCUITPY drive, especially if you get messages complaining of a missing mount point.

Inside the context manager you can access the f variable to operate on the file while it's open. The **write** function is called to write a line of text to the file. Notice that unlike a print statement you need to end the string passed to write with explicit carriage returns and new lines.

You can also open a file and read a line from it with similar code:

```
with open("/sd/test.txt", "r") as f:
    print("Read line from file:")
    print(f.readline())
```

If you wanted to read and print all of the lines from a file you could call **readline** in a loop. Once **readline** reaches the end of the file it will return an empty string so you know when to stop:

```
with open("/sd/test.txt", "r") as f:
    print("Printing lines in file:")
    line = f.readline()
    while line != '':
        print(line)
        line = f.readline()
```

There's even a **readlines** function that will read all of the lines in the file and return them in an array of lines. Be careful though as this means the entire file must be loaded into memory, so if the file is very large you might run out of memory. If you know your file is very small you can use it though:

```
with open("/sd/test.txt", "r") as f:
    lines = f.readlines()
    print("Printing lines in file:")
    for line in lines:
        print(line)
```

Finally one other very common file scenario is opening a file to add new data at the end, or append data. This works exactly the same as in Python and the **open** function can be told you'd like to append instead of erase and write new data (what normally happens with the **w** option for **open**). For example to add a line to the file:

```
with open("/sd/test.txt", "a") as f:
    f.write("This is another line!\r\n")
```

Notice the **a** option in the open function--this tells Python to add data at the end of the file instead of erasing it and starting over at the top. Try reading the file with the code above to see the new line that was added!

That's all there is to manipulating files on microSD cards with CircuitPython!

Here are a few more complete examples of using a SD card from the [Trinket M0 CircuitPython guides (https://adafru.it/Bvi)](https://adafru.it/Bvi). These are great as a reference for more SD card usage.

List Files

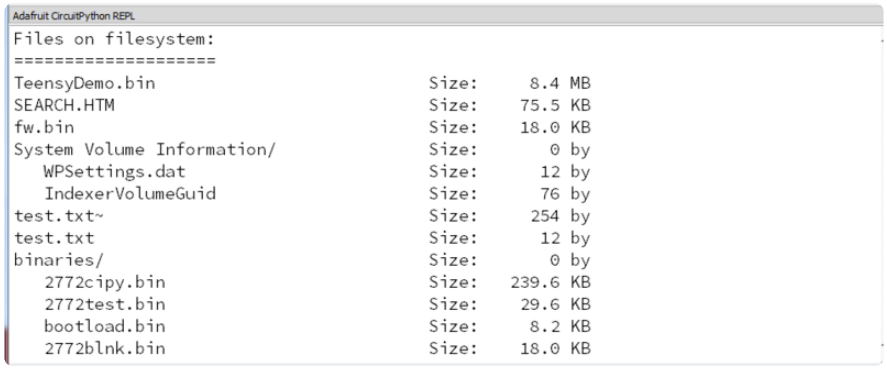
Load this into **code.py** (CircuitPython) or **main.py** (MicroPython):

```
# SPDX-FileCopyrightText: 2017 Limor Fried for Adafruit Industries
#
# SPDX-License-Identifier: MIT

import os

import adafruit_sdcard
import board
import busio
import digitalio
import storage

# Use any pin that is not taken by SPI
SD_CS = board.D0

# Connect to the card and mount the filesystem.
spi = busio.SPI(board.SCK, board.MOSI, board.MISO)
cs = digitalio.DigitalInOut(SD_CS)
sdcard = adafruit_sdcard.SDCard(spi, cs)
vfs = storage.VfsFat(sdcard)
storage.mount(vfs, "/sd")


# Use the filesystem as normal! Our files are under /sd

# This helper function will print the contents of the SD


def print_directory(path, tabs=0):
    for file in os.listdir(path):
        if file == "?":
            continue  # Issue noted in Learn
        stats = os.stat(path + "/" + file)
        filesize = stats[6]
        isdir = stats[0] & 0x4000

        if filesize < 1000:
            sizestr = str(filesize) + " by"
```

```
        elif filesize < 1000000:
            sizestr = "%0.1f KB" % (filesize / 1000)
        else:
            sizestr = "%0.1f MB" % (filesize / 1000000)

        prettyprintname = ""
        for _ in range(tabs):
            prettyprintname += "   "
        prettyprintname += file
        if isdir:
            prettyprintname += "/"
        print('{0:<40} Size: {1:>10}'.format(prettyprintname, sizestr))

        # recursively print directory contents
        if isdir:
            print_directory(path + "/" + file, tabs + 1)


print("Files on filesystem:")
print("====================")
print_directory("/sd")
```

https://github.com/adafruit/Adafruit_Learning_System_Guides/blob/main/Introducing_Trinket_M0/Trinket_SDCardList/code.py

Once it's loaded up, open up the REPL (and restart it with ^D if necessary) to get a printout of all the files included. We recursively print out all files and also the filesize. This is a good demo to start with because you can at least tell if your files exist!



Logging Temperature

But you probably want to do a little more, lets log the temperature from the chip to a file.

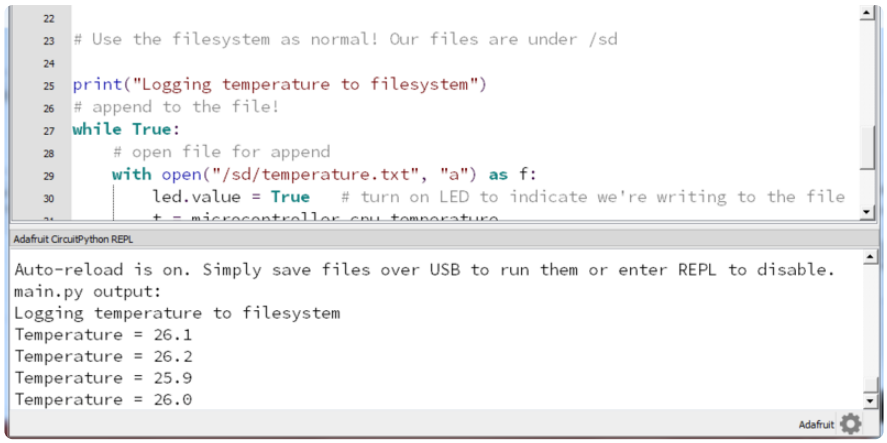
Here's the new script

```
# SPDX-FileCopyrightText: 2017 Limor Fried for Adafruit Industries
#
# SPDX-License-Identifier: MIT

```

```
import time

import adafruit_sdcard
import board
import busio
import digitalio
import microcontroller
import storage

# Use any pin that is not taken by SPI
SD_CS = board.D0

led = digitalio.DigitalInOut(board.D13)
led.direction = digitalio.Direction.OUTPUT

# Connect to the card and mount the filesystem.
spi = busio.SPI(board.SCK, board.MOSI, board.MISO)
cs = digitalio.DigitalInOut(SD_CS)
sdcard = adafruit_sdcard.SDCard(spi, cs)
vfs = storage.VfsFat(sdcard)
storage.mount(vfs, "/sd")

# Use the filesystem as normal! Our files are under /sd

print("Logging temperature to filesystem")
# append to the file!
while True:
    # open file for append
    with open("/sd/temperature.txt", "a") as f:
        led.value = True  # turn on LED to indicate we're writing to the file
        t = microcontroller.cpu.temperature
        print("Temperature = %0.1f" % t)
        f.write("%0.1f\n" % t)
        led.value = False  # turn off LED to indicate we're done
    # file is saved
    time.sleep(1)
```

https://github.com/adafruit/Adafruit_Learning_System_Guides/blob/main/Introducing_Trinket_M0/Trinket_SDlogger/code.py

When saved, the Trinket will start saving the temperature once per second to the SD card under the file **temperature.txt**



The key part of this demo is in these lines:

```
print("Logging temperature to filesystem")
# append to the file!
while True:
    # open file for append
    with open("/sd/temperature.txt", "a") as f:
        led.value = True    # turn on LED to indicate we're writing to the file
        t = microcontroller.cpu.temperature
        print("Temperature = %0.1f" % t)
        f.write("%0.1f\n" % t)
        led.value = False    # turn off LED to indicate we're done
    # file is saved
    time.sleep(1)
```

This is a slightly complex demo but it's for a good reason. We use **with** (a 'context') to open the file for **a**ppending, that way the file is only opened for the very short time its written to. This is safer because then if the SD card is removed or the board turned off, all the data will be safe(r).

The LED is used to let the person using this know that the temperature is being written, it turns on just before the write and then off right after.

After the LED is turned off the **with** ends and the context closes, the file is safely stored.

Python/Linux

This breakout is not for use with single board linux computers like Raspberry Pi, etc. If you want to add an SD card to a Raspberry Pi or other computer, please use a USB to SD card adapter like this one:

Featured

USB MicroSD Card Reader/Writer - microSD / microSDHC / microSDXC

This is the cutest little microSD card reader/writer - but don't be fooled by its adorableness! It's wicked fast and supports up to 64 GB SDXC cards! Simply slide the card into...

https://www.adafruit.com/product/939

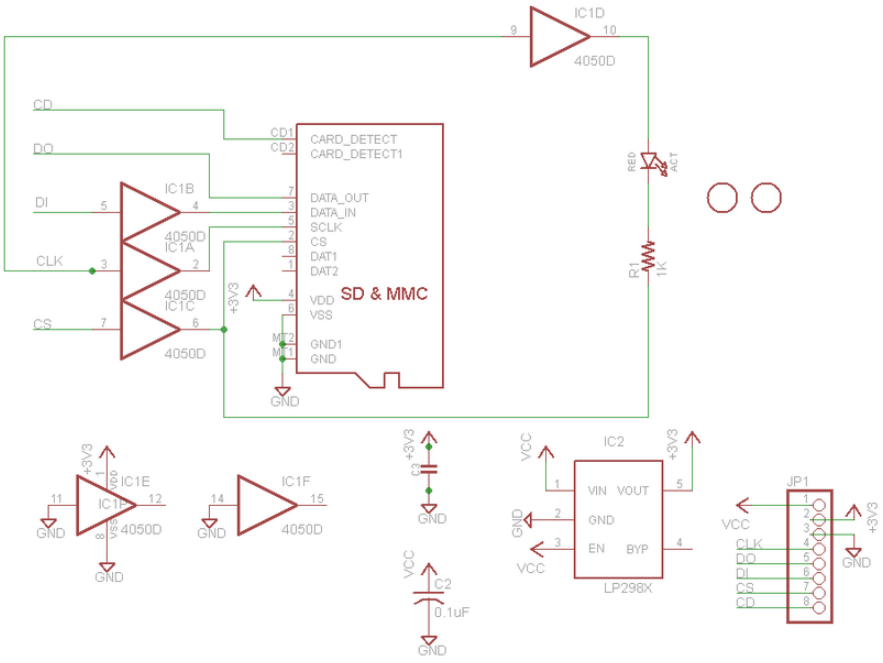
This board is for use with microcontrollers only!

Download

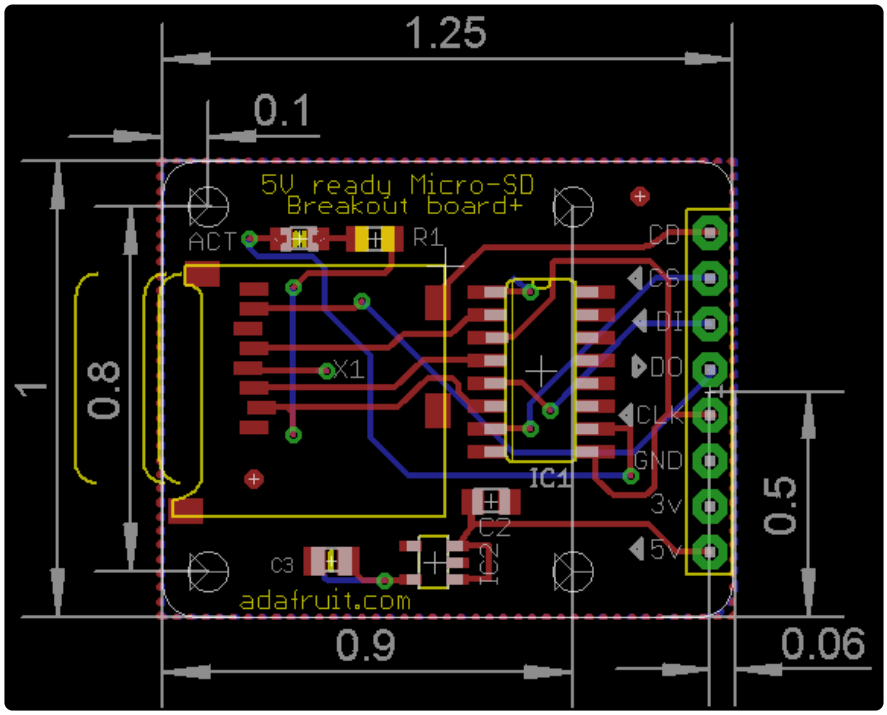
- Transcend microSD card datasheet (https://adafru.it/cma)
- EagleCAD PCB files on GitHub (https://adafru.it/rfT)
- Fritzing object in the Adafruit Fritzing library (https://adafru.it/aP3)

Schematic

Click to embiggen. Same schematic for Arduino and CircuitPython (same hardware).



For the level shifter we use the CD74HC4050 (https://adafru.it/Boj) which has a typical propagation delay of ~10ns

Fabrication Print

Dims in inches