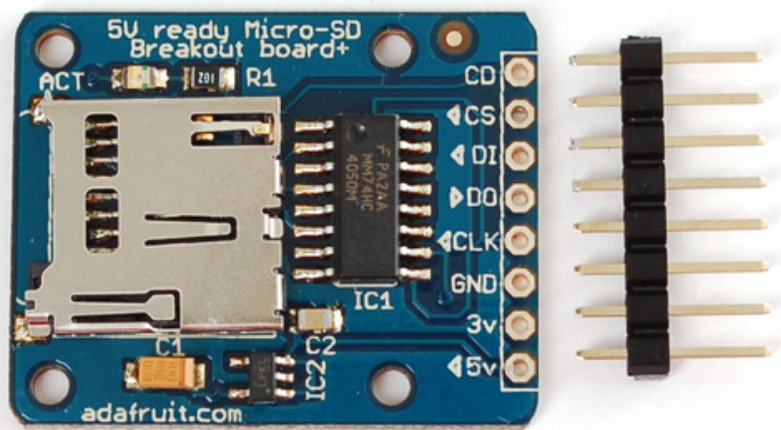


Micro SD Card Breakout Board Tutorial

Created by lady ada

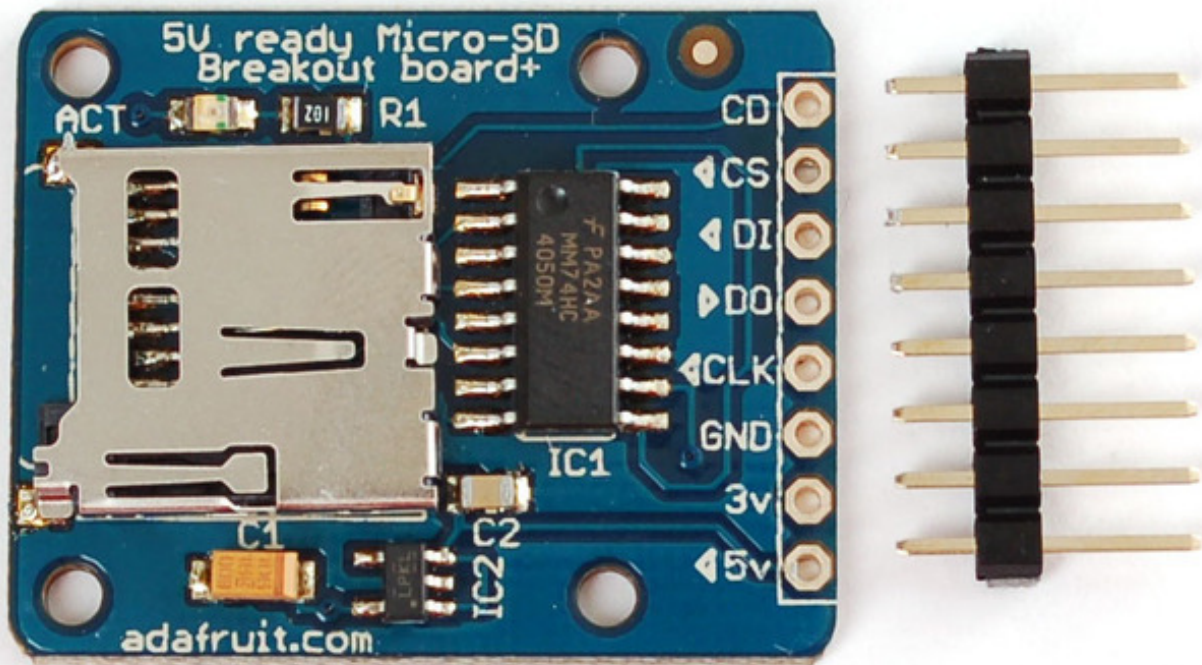


Last updated on 2014-05-07 04:30:09 PM EDT

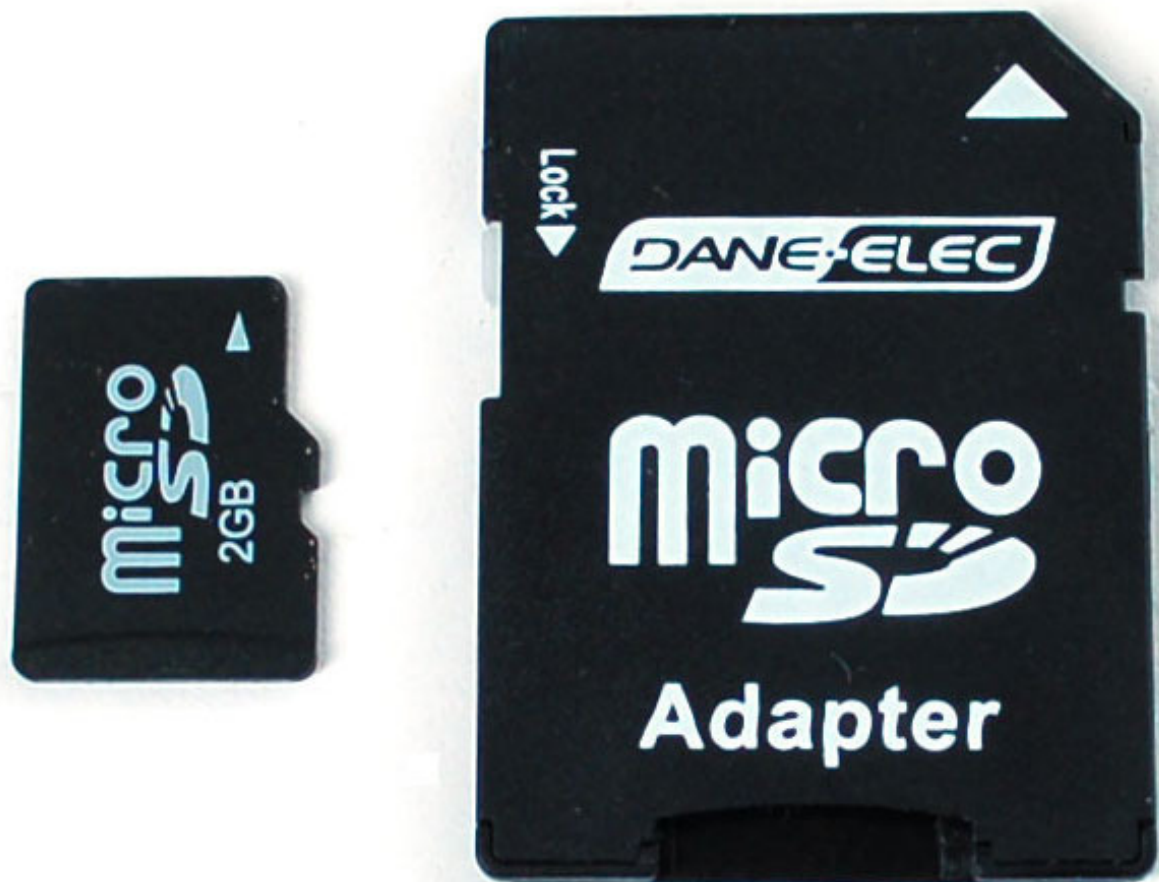
Guide Contents

Guide Contents	2
Introduction	3
Look out!	6
What to watch for!	6
Formatting notes	7
Wiring	8
Library	10
Arduino Library & First Test	10
Reading files	15
Reading from files	17
Recursively listing/reading files	18
Functions	20
Other useful functions	20
Examples	21
More examples!	21
Download	22

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.

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 card. 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.

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!

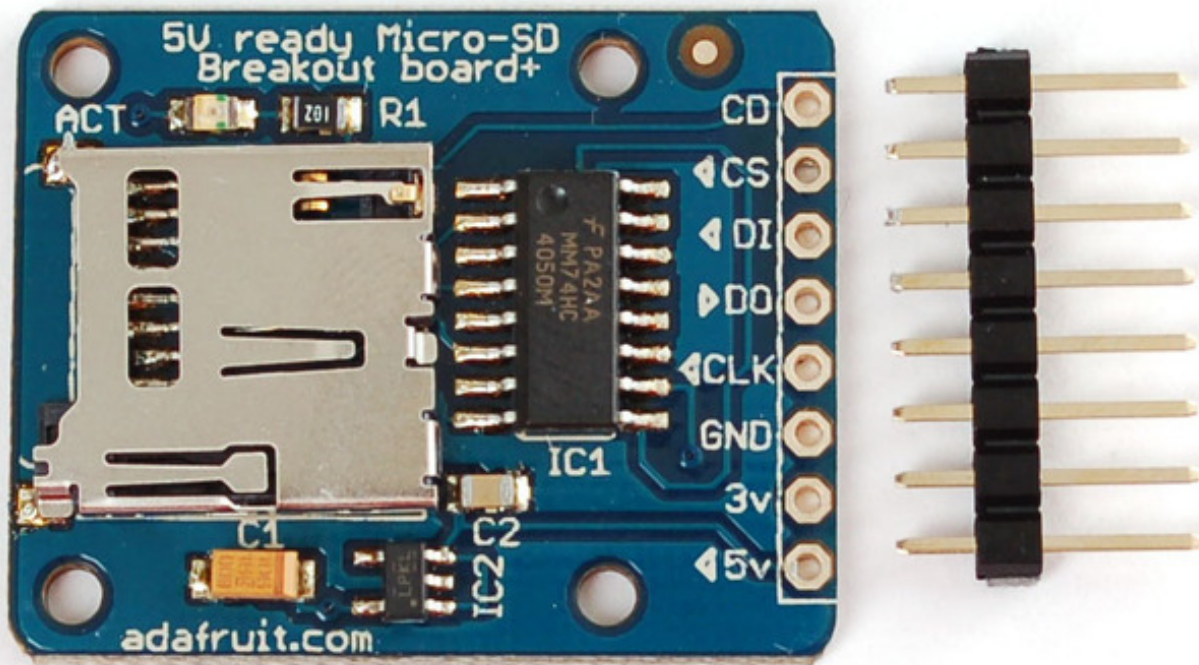
Download the formatter from

https://www.sdcard.org/downloads/formatter_3/ (<http://adafru.it/c73>)

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

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.



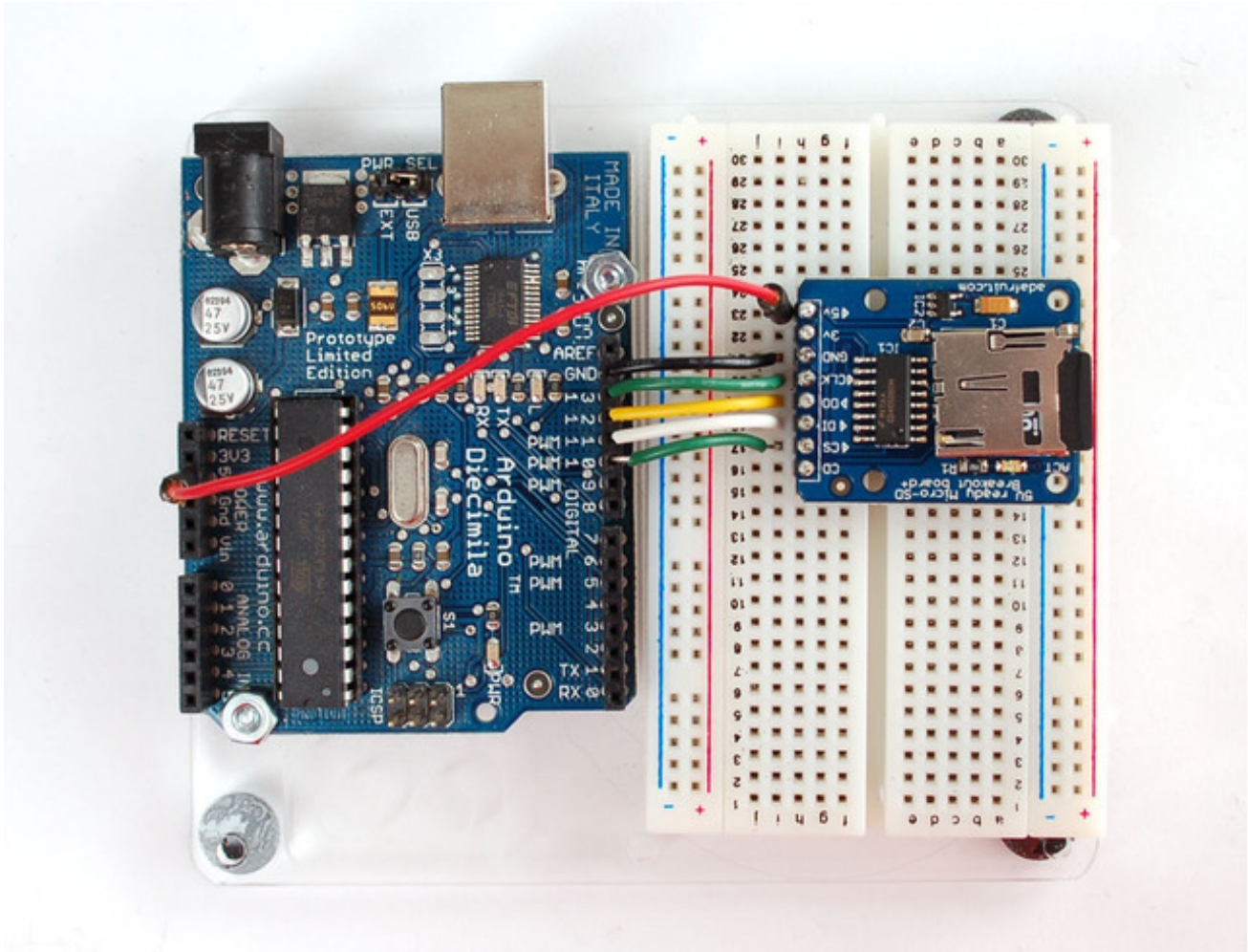
In this tutorial we will be using an Arduino to demonstrate the wiring and interfacing. If you have another microcontroller, you'll need to adapt the wiring and code to match!

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/slave 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 inserted. 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!

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?)

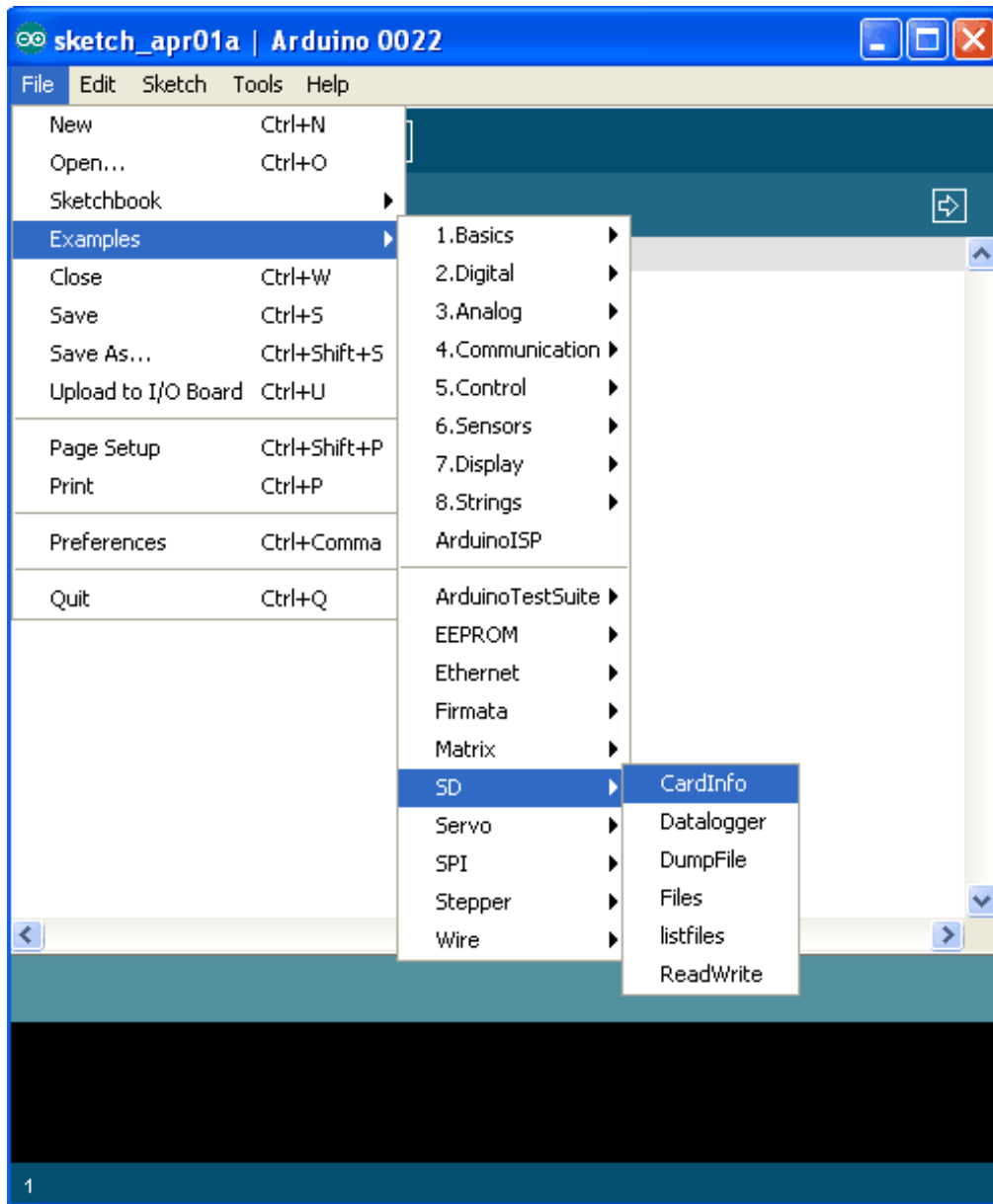
If you want to use the bleeding edge latest version of SD which includes some things that are not wrapped up into the IDE such as being able to use any pins not just hardware SPI, you can update to the Adafruit version of SD: [download the library from our Github repository \(http://adafru.it/aP6\)](http://adafru.it/aP6) by clicking the **DOWNLOAD** button below.

Download Adafruit's SD Library

<http://adafru.it/cxl>

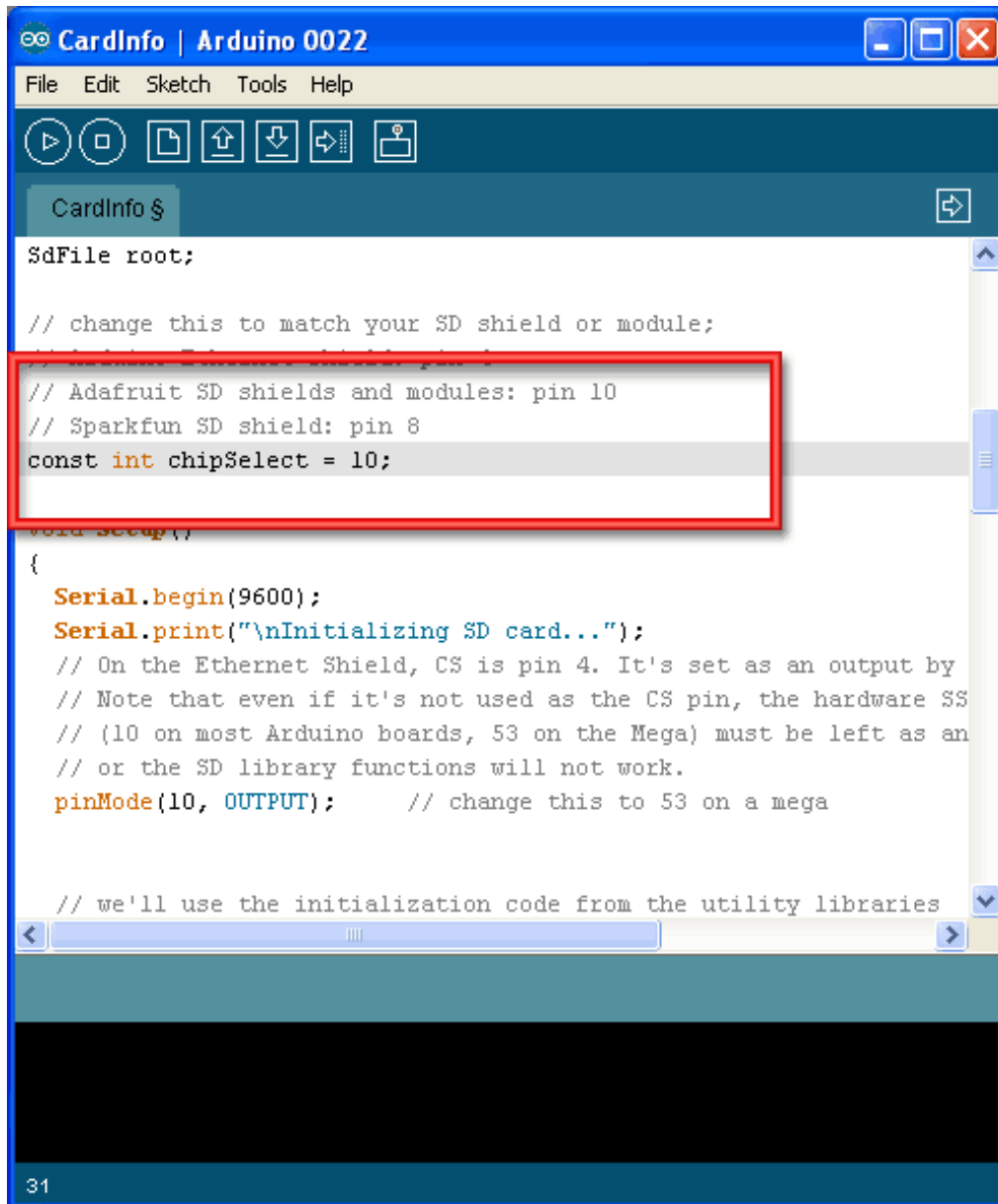
Then make a backup of the folder called **SD** in your **ArduinoIDE/libraries** folder (on a Mac you will have to 'explore' the App). Then uncompress the newly downloaded folder and rename it **SD**. Inside the **SD** folder you should see **README.txt** and other files. Install it by dragging it in your **ArduinoIDE/libraries** folder and restarting the IDE

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!



The screenshot shows the Arduino IDE interface with the 'CardInfo' sketch loaded. The title bar reads 'CardInfo | Arduino 0022'. The menu bar includes 'File', 'Edit', 'Sketch', 'Tools', and 'Help'. Below the menu bar is a toolbar with icons for running, stopping, saving, and other functions. The main text area contains the following code:

```
CardInfo $
SdFile root;

// change this to match your SD shield or module;
// Sparkfun SD shield: pin 8
// Adafruit SD shields and modules: pin 10
const int chipSelect = 10;

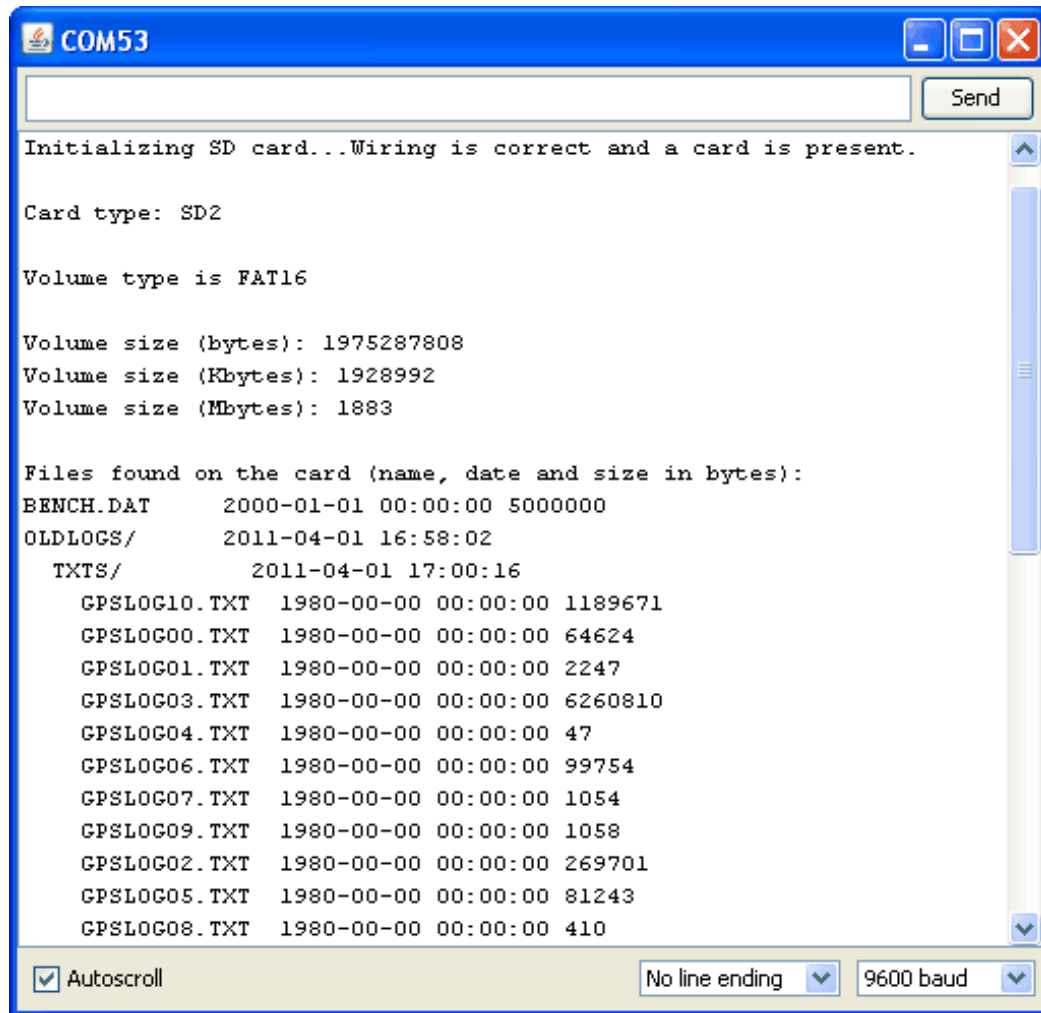
void setup()
{
  Serial.begin(9600);
  Serial.print("\nInitializing SD card...");
  // On the Ethernet Shield, CS is pin 4. It's set as an output by
  // Note that even if it's not used as the CS pin, the hardware SS
  // (10 on most Arduino boards, 53 on the Mega) must be left as an
  // or the SD library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

  // we'll use the initialization code from the utility libraries
```

A red rectangular box highlights the lines: `// Adafruit SD shields and modules: pin 10` and `const int chipSelect = 10;`. The bottom status bar shows the line number '31'.

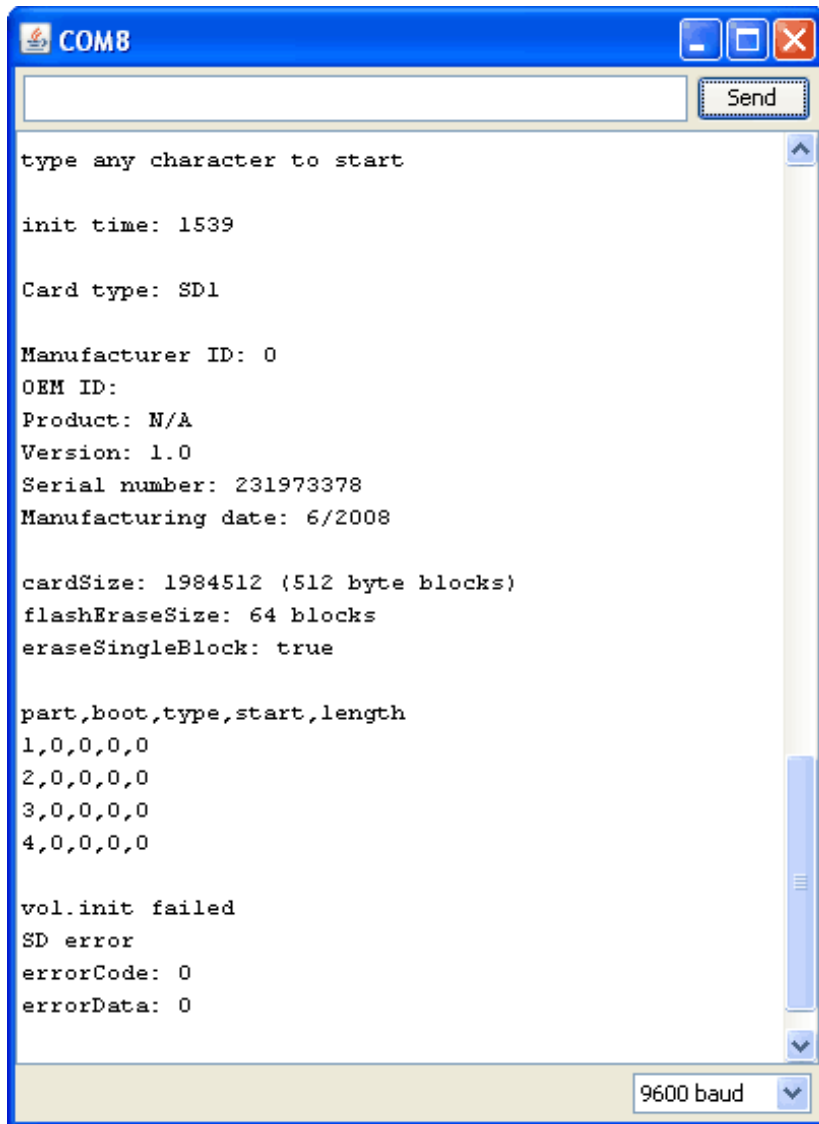
OK, now insert the SD card into the breakout board and upload the sketch.

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:



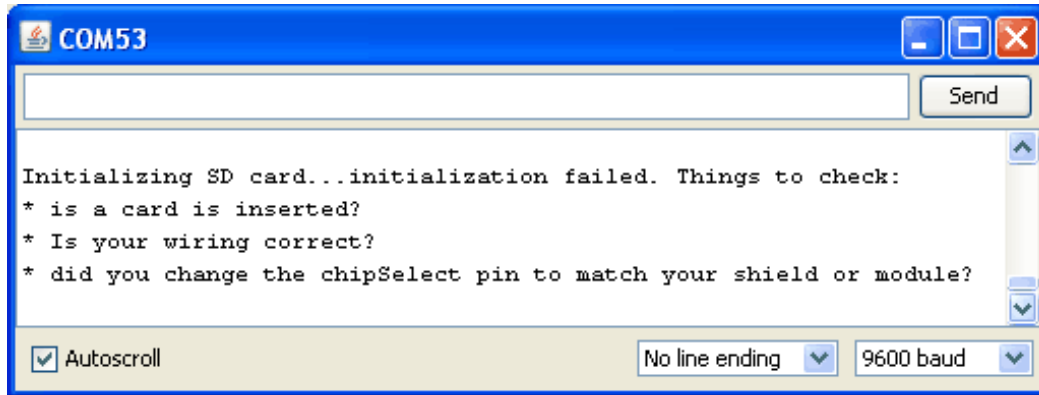
It's mostly gibberish, but it's useful to see the **Volume type is FAT16** part as well as the size of the card (about 2 GB which is what it should be) etc.

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.

Reading 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) {
```

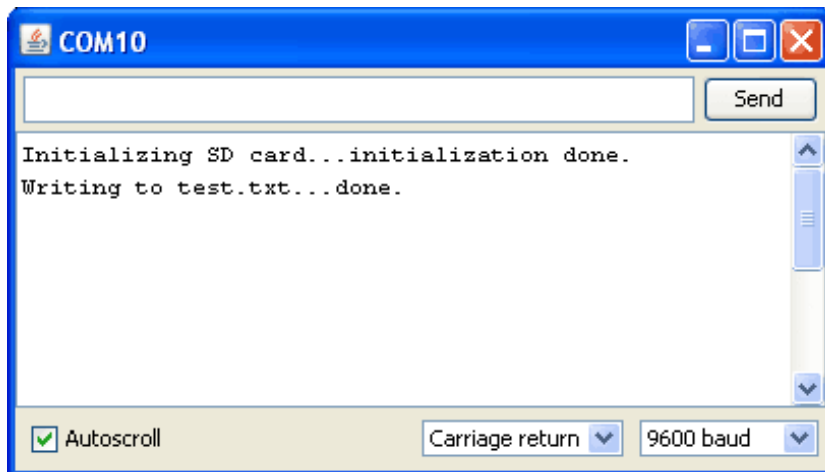
```

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:
  Serial.println("error opening test.txt");
}
}

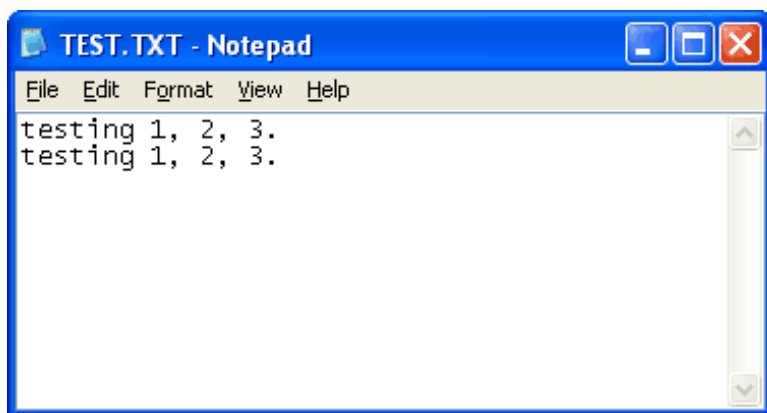
void loop()
{
  // nothing happens after setup
}

```

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.

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

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.");

  // open the file for reading:
```

```

myFile = SD.open("test.txt");
if (myFile) {
  Serial.println("test.txt:");

  // read from the file until there's nothing else in it:
  while (myFile.available()) {
    Serial.write(myFile.read());
  }
  // close the file:
  myFile.close();
} else {
  // if the file didn't open, print an error:
  Serial.println("error opening test.txt");
}
}

void loop()
{
  // nothing happens after setup
}

```

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.

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

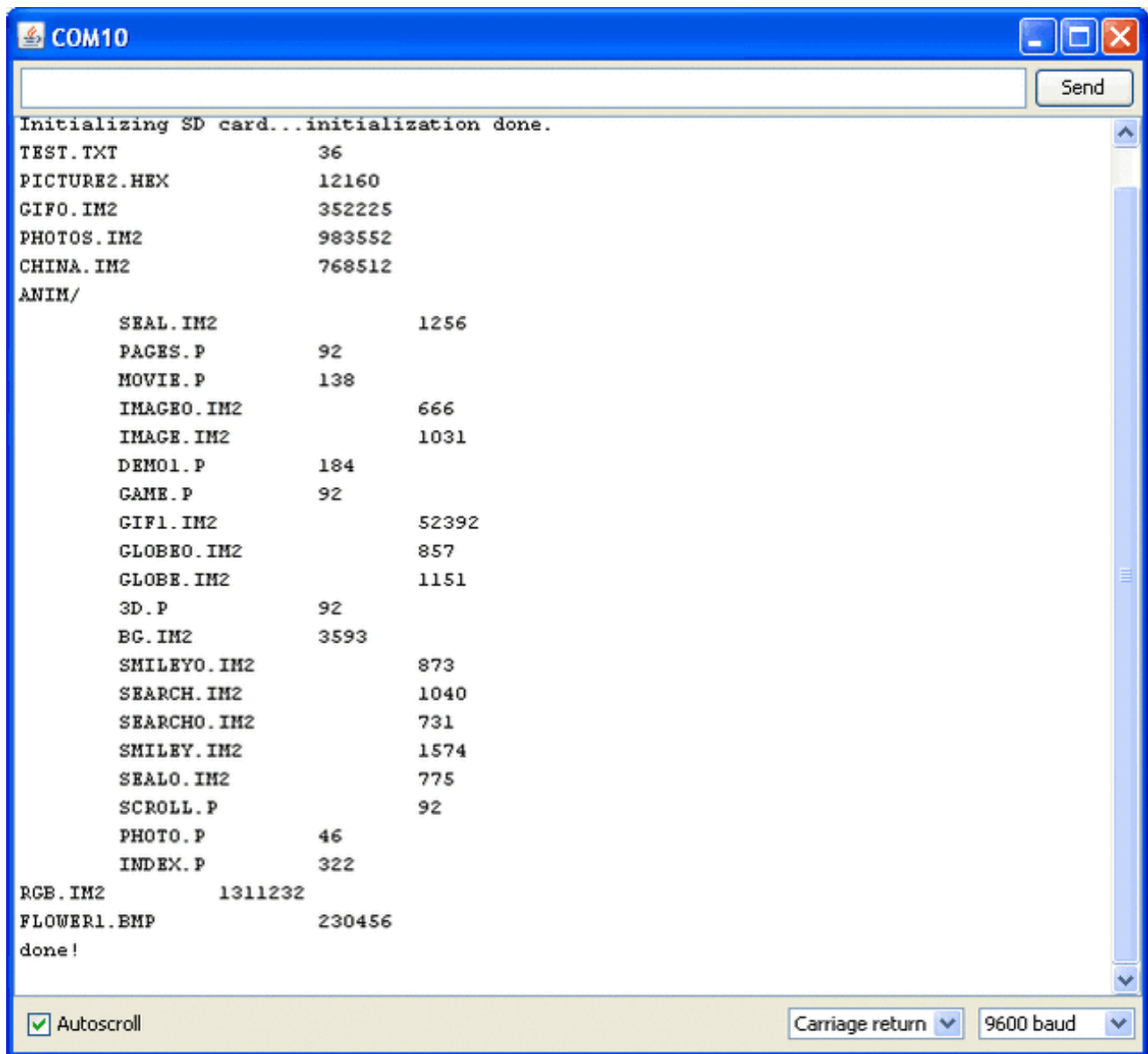
Recursively listing/reading files

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.



The screenshot shows a serial monitor window titled "COM10". It displays the output of a program that has initialized an SD card and listed its contents. The output shows a list of files and their sizes in bytes. The files are listed in a hierarchical manner, with some files grouped under a directory named "ANIM/". The window includes a "Send" button at the top right and a status bar at the bottom with options for "Autoscroll", "Carriage return", and "9600 baud".

```
Initializing SD card...initialization done.  
TEST.TXT          36  
PICTURE2.HEX      12160  
GIF0.IM2          352225  
PHOTOS.IM2        983552  
CHINA.IM2         768512  
ANIM/  
    SEAL.IM2      1256  
    PAGES.P       92  
    MOVIE.P       138  
    IMAGE0.IM2    666  
    IMAGE.IM2     1031  
    DEMO1.P       184  
    GAME.P        92  
    GIF1.IM2      52392  
    GLOBE0.IM2    857  
    GLOBE.IM2     1151  
    3D.P          92  
    BG.IM2        3593  
    SMILEY0.IM2   873  
    SEARCH.IM2    1040  
    SEARCH0.IM2   731  
    SMILEY.IM2    1574  
    SEAL0.IM2     775  
    SCROLL.P      92  
    PHOTO.P       46  
    INDEX.P       322  
RGB.IM2           1311232  
FLOWER1.BMP       230456  
done!
```

Functions

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 \(http://adafruit.it/dpH\)](http://adafruit.it/dpH) and [GPS logging shield \(http://adafruit.it/dpI\)](http://adafruit.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 \(http://adafruit.it/dpJ\)](http://adafruit.it/dpJ) and [1.8" TFT breakout tutorials \(http://adafruit.it/ckK\)](http://adafruit.it/ckK). Those have examples of how we read BMP files off disk and parse them.

Download

[Transcend microSD card datasheet \(http://adafru.it/cma\)](http://adafru.it/cma)

Our latest SD card library version (<http://adafru.it/aP6>)- download it by clicking DOWNLOADS at the top right. Then make a backup of the folder called **SD** in your **ArduinoIDE/libraries** folder (on a Mac you will have to 'explore' the App). Then uncompress the newly downloaded folder and rename it **SD**. Inside the **SD** folder you should see **README.txt** and other files. Install it by dragging it in your **ArduinoIDE/libraries** folder and restarting the IDE.