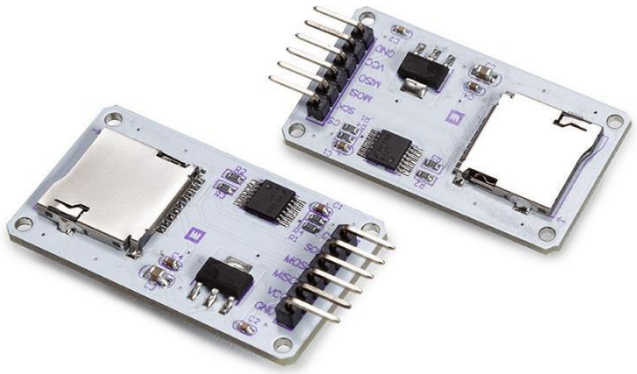
WHADDA
EXCITING ELECTRONICS

MANUAL

EN microSD Card Logging Shield for Arduino®

WPI304N



whadda.com

Introduction

To all residents of the European Union
Important environmental information about this product
This symbol on the device or the package indicates that disposal of the device after its lifecycle could harm the environment. Do not dispose of the unit (or batteries) as unsorted municipal waste; it should be taken to a specialized company for recycling. This device should be returned to your distributor or to a local recycling service. Respect the local environmental rules.
If in doubt, contact your local waste disposal authorities.

Thank you for choosing Whadda! Please read the manual thoroughly before bringing this device into service. If the device was damaged in transit, do not install or use it and contact your dealer.

Safety Instructions

Read and understand this manual and all safety signs before using this appliance.

For indoor use only.

- This device can be used by children aged from 8 years and above, and persons with reduced physical, sensory or mental capabilities or lack of experience and knowledge if they have been given supervision or instruction concerning the use of the device in a safe way and understand the hazards involved. Children shall not play with the device. Cleaning and user maintenance shall not be made by children without supervision.

General Guidelines

- Refer to the Velleman® Service and Quality Warranty on the last pages of this manual.
- All modifications of the device are forbidden for safety reasons. Damage caused by user modifications to the device is not covered by the warranty.
- Only use the device for its intended purpose. Using the device in an unauthorized way will void the warranty.
- Damage caused by disregard of certain guidelines in this manual is not covered by the warranty and the dealer will not accept responsibility for any ensuing defects or problems.
- Nor Velleman Group nv nor its dealers can be held responsible for any damage (extraordinary, incidental or indirect) – of any nature (financial, physical…) arising from the possession, use or failure of this product.
- Keep this manual for future reference.

What is Arduino®

Arduino® is an open-source prototyping platform based on easy-to-use hardware and software. Arduino® boards are able to read inputs – light-on sensor, a finger on a button or a Twitter message – and turn it into an output – activating of a motor, turning on an LED, publishing something online. You can tell your board what to do by sending a set of instructions to the microcontroller on the board. To do so, you use the Arduino programming language (based on Wiring) and the Arduino® software IDE (based on Processing). Additional shields/modules/components are required for reading a twitter message or publishing online. Surf to www.arduino.cc for more information.

Product overview

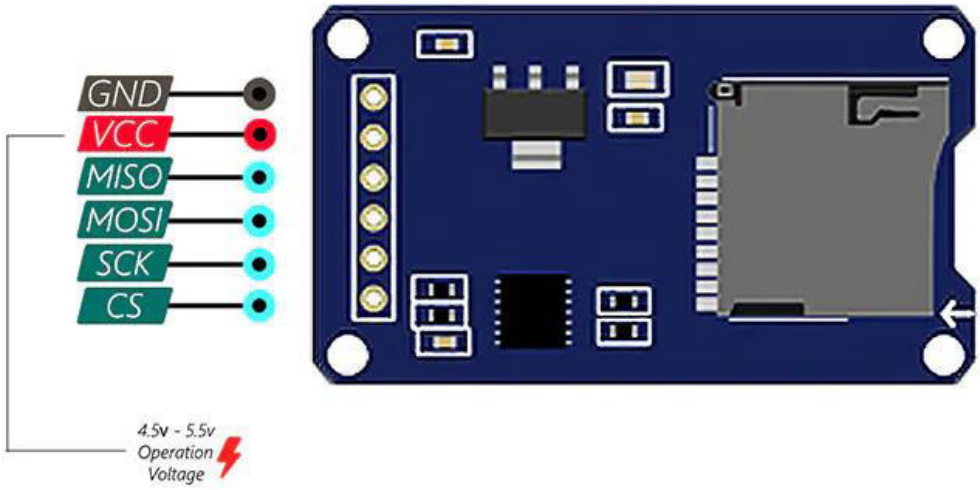
This shield will prove useful for data logging with your Arduino®. Can be easily assembled and customized for any data-logging project.
You can use this card to access microSD memory cards using SPI protocol in your microcontroller projects.

Specifications

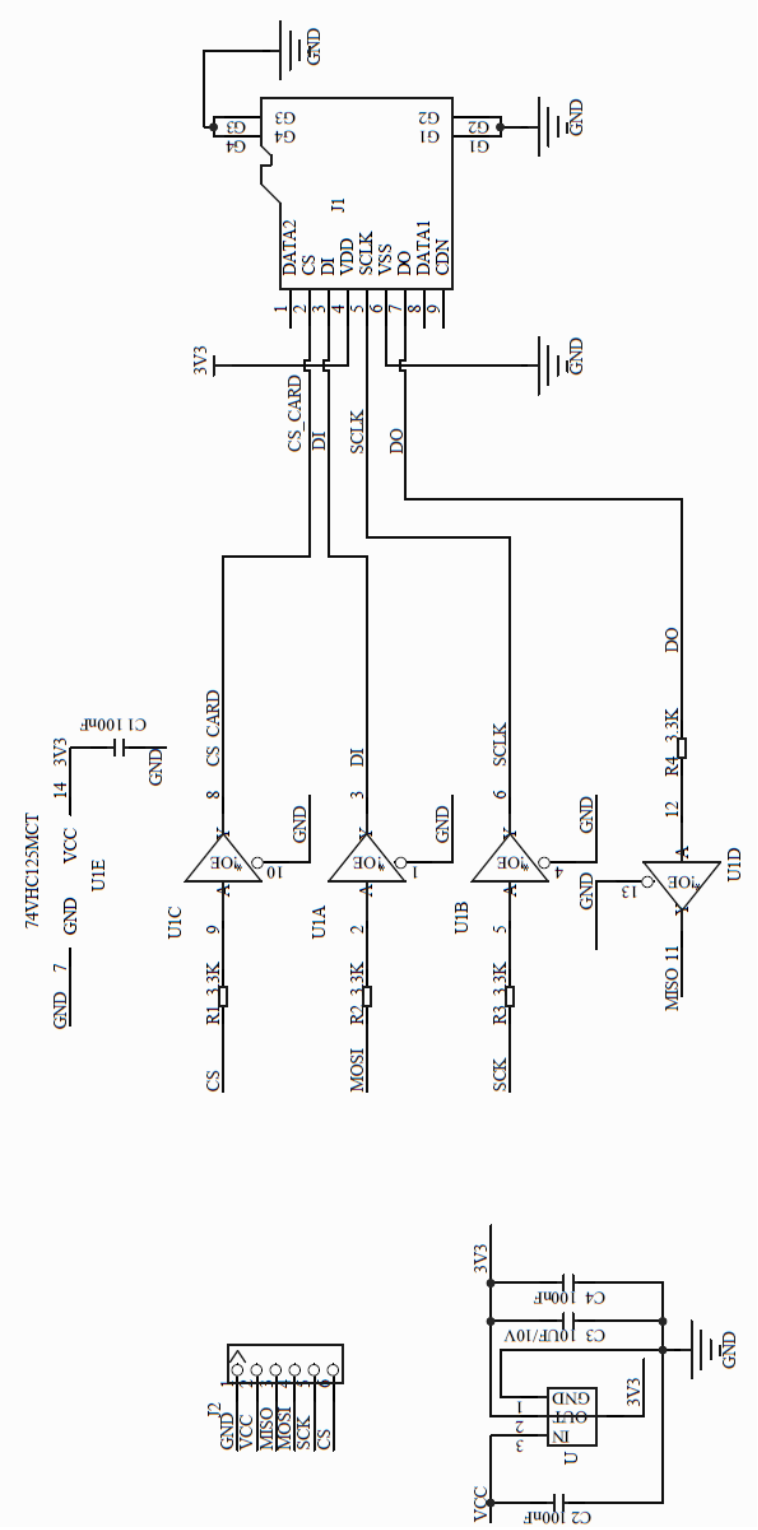
- supports microSD cards (≤ 2 GB) and microSDHC cards (≤ 32 GB) (high-speed)
- onboard voltage level conversion circuit that interfaces the data voltages between 5 V from Arduino® controller and 3.3 V to SD card data pins
- power supply: 4.5-5.5 V
- onboard voltage regulator 3V3, for voltage level circuit
- communication interface: SPI bus
- 4x M2 screw positioning holes for easy installation
- size: 4.1 x 2.4 cm

Wiring

Logging shield	To Arduino® Uno	To Arduino® Mega
CS (cable select)	4	53
SCK (CLK)	13	52
MOSI	11	51
MISO	12	50
5V (4.5V-5.5V)	5V	5V
GND	GND	GND



Circuit Diagram



Operation

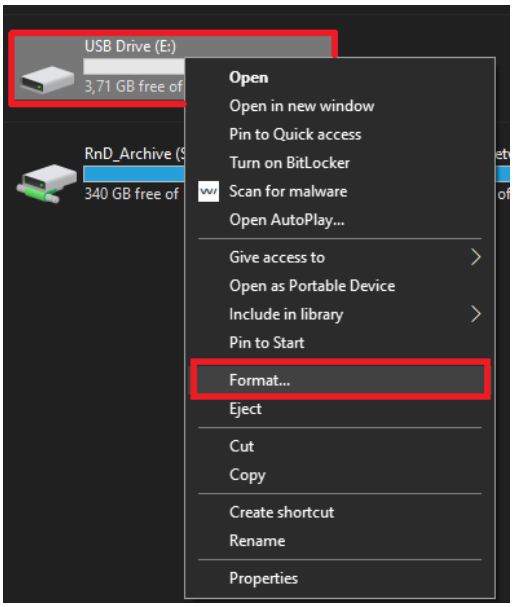
Introduction

The WPI304N SD card module is especially useful for projects that require data logging. Arduino® can create a file onto an SD card to write and save data, using the standard **SD** library from Arduino® IDE. The WPI304N module uses the SPI communication protocol.

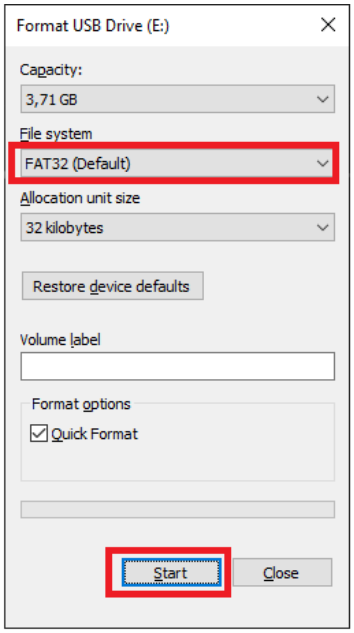
Preparing the microSD card

The first step when using the WPI304N SD card module with Arduino®, is formatting the microSD card as a FAT16 or FAT32 file system. Follow the instructions below:

1. Insert the SD card in your computer. Go to **My Computer** and right-click on the SD card removable drive. Select **Format** as shown in picture below.

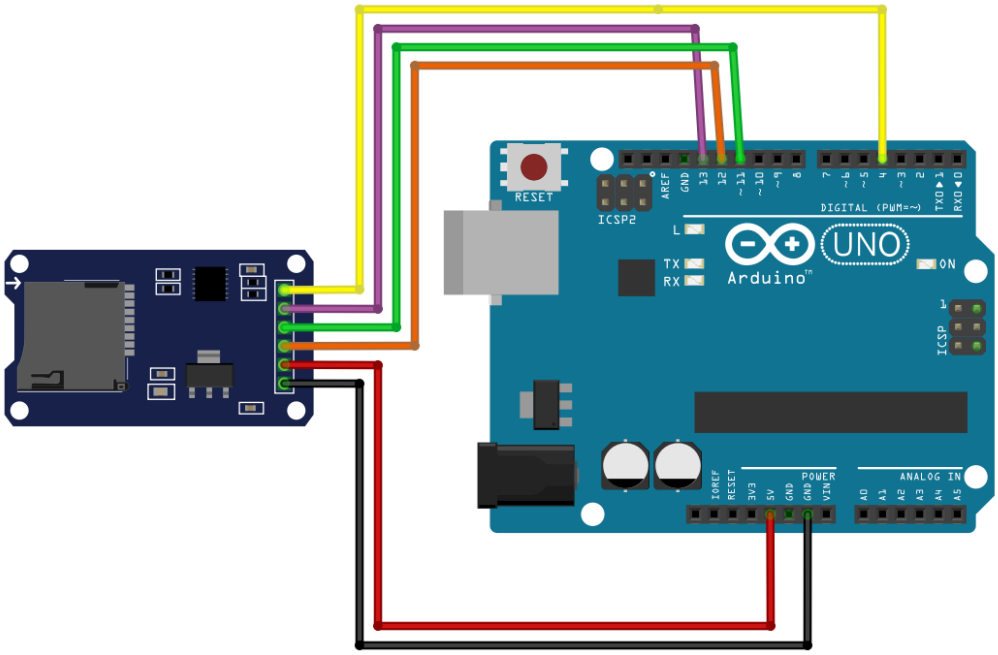


2. A new window pops up. Select **FAT32**, press **Start** to initialize the formatting process and follow the onscreen instructions.



Using the SD card module

Insert the formatted microSD card in the SD card module. Connect the SD card module to the Arduino® Uno as shown in the circuit below, or check the pin assignment table in a previous section.

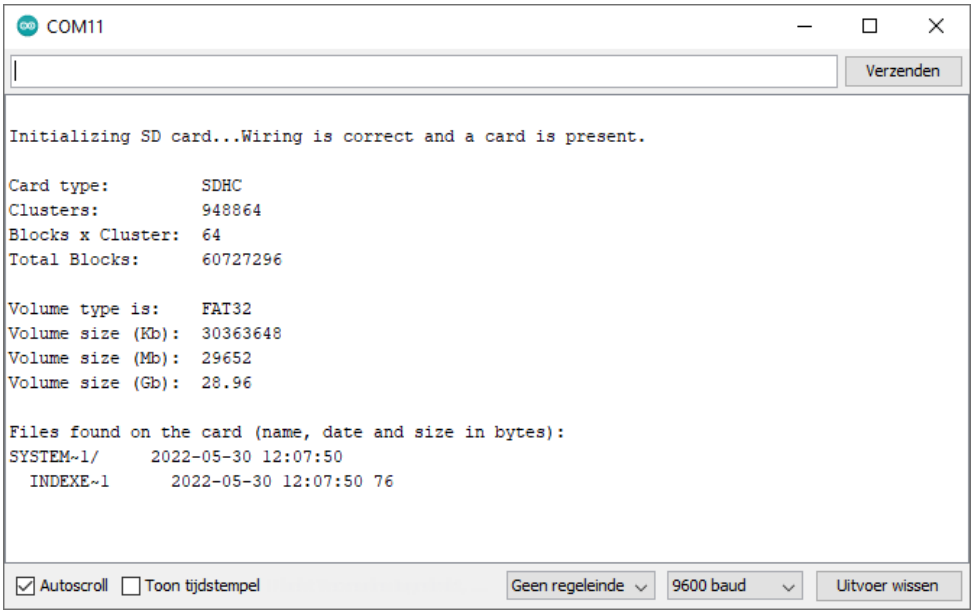


Coding

SD card info

To make sure everything is wired correctly, and the SD card is working, go to **File → Examples → SD → CardInfo** in the Arduino® IDE software.

Now, upload the code to your Arduino® Uno board. Make sure to select the right board and COM port. Open the serial monitor with baud rate **9600**. Normally, your microSD card information will be presented in the serial monitor. If everything is working properly, you will see a similar message on the serial monitor.



Reading and writing data onto the microSD card

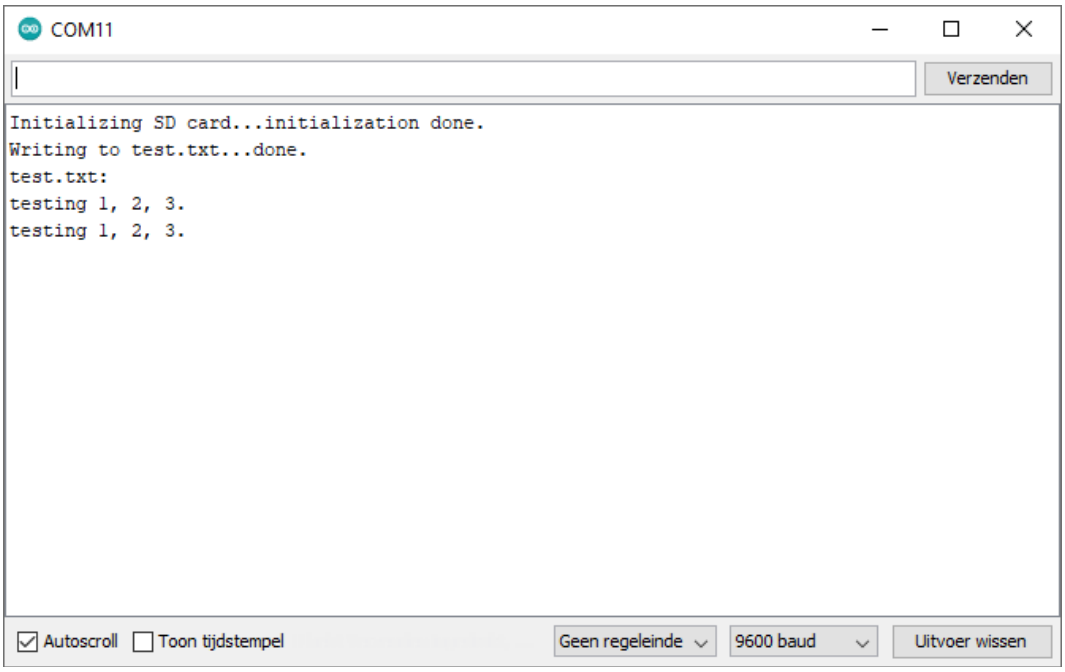
The SD library provides useful functions which allows to easily write onto and read from an SD card. Open the **ReadWrite** example from **File → Examples → SD → ReadWrite** and upload it to your Arduino® Uno board.

Code

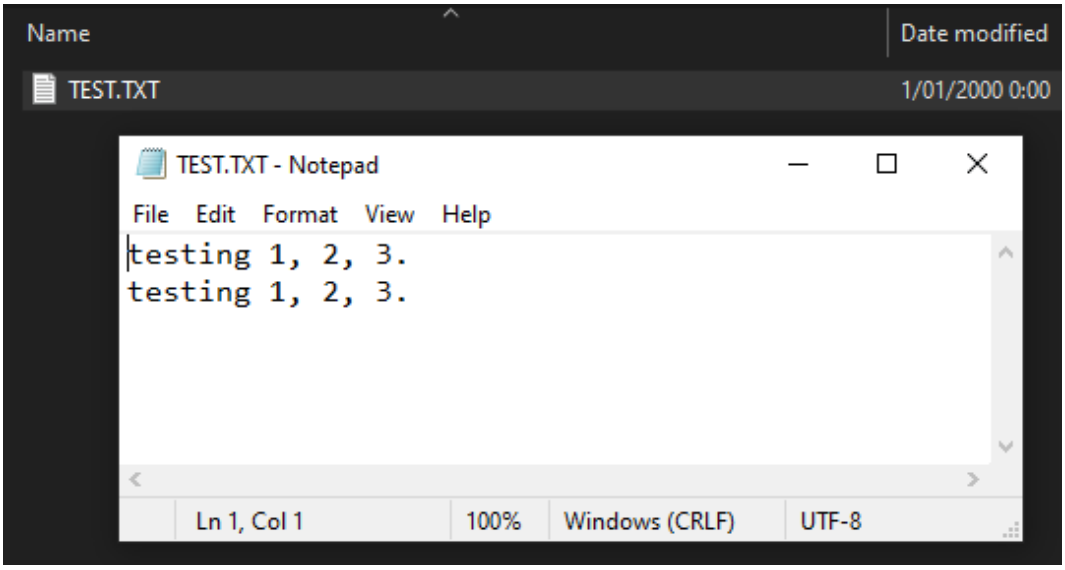
```
/*
  SD card read/write

  This example shows how to read and write data to and from an SD card file
  The circuit:
   SD card attached to SPI bus as follows:
 ** MOSI - pin 11
 ** MISO - pin 12
 ** CLK - pin 13
 ** CS - pin 4 (for MKRZero SD: SDCARD_SS_PIN)

  created   Nov 2010
  by David A. Mellis
  modified 9 Apr 2012
  by Tom Igoe

  This example code is in the public domain.

*/

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

File myFile;

void setup() {
  // Open serial communications and wait for port to open:
```

```
28.   Serial.begin(9600);
29.   while (!Serial) {
30.     ; // wait for serial port to connect. Needed for native USB port only
31.   }
32.
33.
34.   Serial.print("Initializing SD card...");
35.
36.   if (!SD.begin(4)) {
37.     Serial.println("initialization failed!");
38.     while (1);
39.   }
40.   Serial.println("initialization done.");
41.
42.   // open the file. note that only one file can be open at a time,
43.   // so you have to close this one before opening another.
44.   myFile = SD.open("test.txt", FILE_WRITE);
45.
46.   // if the file opened okay, write to it:
47.   if (myFile) {
48.     Serial.print("Writing to test.txt...");
49.     myFile.println("testing 1, 2, 3.");
50.     // close the file:
51.     myFile.close();
52.     Serial.println("done.");
53.   } else {
54.     // if the file didn't open, print an error:
55.     Serial.println("error opening test.txt");
56.   }
57.
58.   // re-open the file for reading:
59.   myFile = SD.open("test.txt");
60.   if (myFile) {
61.     Serial.println("test.txt:");
62.
63.     // read from the file until there's nothing else in it:
64.     while (myFile.available()) {
65.       Serial.write(myFile.read());
66.     }
67.     // close the file:
68.     myFile.close();
69.   } else {
70.     // if the file didn't open, print an error:
71.     Serial.println("error opening test.txt");
72.   }
73. }
74.
75. void loop() {
76.   // nothing happens after setup
77. }
```

Once the code is uploaded and everything is okay, the following window appears on the serial monitor.



This indicates **reading/writing** was successful. To check about the files on the SD card, use Notepad to open the **TEST.TXT** file on the microSD card. Following data appears in **.txt format**:



NonBlockingWrite.ino example

In the original example NonBlockingWrite code, change line 48

```
if (!SD.begin()) {
```

to

```
if (!SD.begin(4)) {
```

Also, add following lines after line 84:

```
// print the buffer length. This will change depending on when
// data is actually written to the SD card file:
Serial.print("Unsaved data buffer length (in bytes): ");
Serial.println(buffer.length());
// note the time that the last line was added to the string
```

The complete code should be as follows:

```
1.  /*
2.    Non-blocking Write
3.
4.    This example demonstrates how to perform non-blocking writes
5.    to a file on a SD card. The file will contain the current millis()
6.    value every 10ms. If the SD card is busy, the data will be buffered
7.    in order to not block the sketch.
8.
9.    NOTE: myFile.availableForWrite() will automatically sync the
10.         file contents as needed. You may lose some unsynced data
11.         still if myFile.sync() or myFile.close() is not called.
12.
13.   The circuit:
14.   SD card attached to SPI bus as follows:
15.   MOSI - pin 11
16.   MISO - pin 12
17.   SCK / CLK - pin 13
18.   CS - pin 4 (for MKRZero SD: SDCARD_SS_PIN)
19.
20.   This example code is in the public domain.
21. */
22.
23. #include <SD.h>
24.
25. // file name to use for writing
26. const char filename[] = "demo.txt";
27.
28. // File object to represent file
29. File txtFile;
30.
31. // string to buffer output
```

```
32. String buffer;
33.
34. unsigned long lastMillis = 0;
35.
36. void setup() {
37.   Serial.begin(9600);
38.   while (!Serial);
39.   Serial.print("Initializing SD card...");
40.
41.   // reserve 1kB for String used as a buffer
42.   buffer.reserve(1024);
43.
44.   // set LED pin to output, used to blink when writing
45.   pinMode(LED_BUILTIN, OUTPUT);
46.
47.   // init the SD card
48.   if (!SD.begin(4)) {
49.     Serial.println("Card failed, or not present");
50.     Serial.println("initialization failed. Things to check:");
51.     Serial.println("1. is a card inserted?");
52.     Serial.println("2. is your wiring correct?");
53.     Serial.println("3. did you change the chipSelect pin to match your shield or
    module?");
54.     Serial.println("Note: press reset button on the board and reopen this Serial Monitor
    after fixing your issue!");
55.     // don't do anything more:
56.     while (1);
57.   }
58.
59.   // If you want to start from an empty file,
60.   // uncomment the next line:
61.   // SD.remove(filename);
62.
63.   // try to open the file for writing
64.   txtFile = SD.open(filename, FILE_WRITE);
65.   if (!txtFile) {
66.     Serial.print("error opening ");
67.     Serial.println(filename);
68.     while (1);
69.   }
70.
71.   // add some new lines to start
72.   txtFile.println();
73.   txtFile.println("Hello World!");
74.   Serial.println("Starting to write to file...");
75. }
76.
77. void loop() {
78.   // check if it's been over 10 ms since the last line added
79.   unsigned long now = millis();
80.   if ((now - lastMillis) >= 10) {
81.     // add a new line to the buffer
82.     buffer += "Hello ";
83.     buffer += now;
84.     buffer += "\r\n";
```

```
85.     // print the buffer length. This will change depending on when
86.     // data is actually written to the SD card file:
87.     Serial.print("Unsaved data buffer length (in bytes): ");
88.     Serial.println(buffer.length());
89.     // note the time that the last line was added to the string
90.     lastMillis = now;
91.   }
92.
93.   // check if the SD card is available to write data without blocking
94.   // and if the buffered data is enough for the full chunk size
95.   unsigned int chunkSize = txtFile.availableForWrite();
96.   if (chunkSize && buffer.length() >= chunkSize) {
97.     // write to file and blink LED
98.     digitalWrite(LED_BUILTIN, HIGH);
99.     txtFile.write(buffer.c_str(), chunkSize);
100.     digitalWrite(LED_BUILTIN, LOW);
101.
102.     // remove written data from buffer
103.     buffer.remove(0, chunkSize);
104.   }
105. }
```

whadda.com