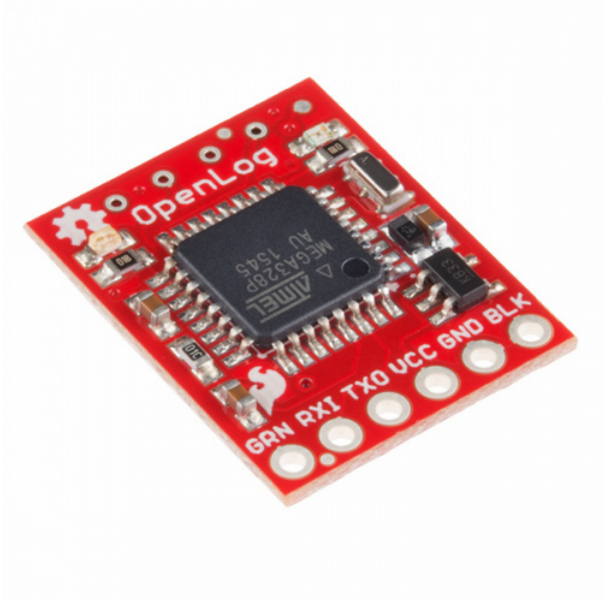
OpenLog Hookup Guide

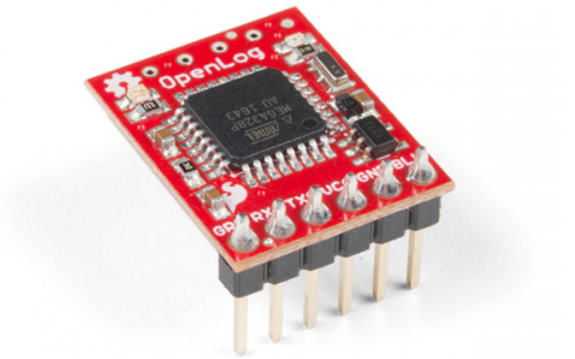
Introduction

Heads up! This tutorial is for the Open Log for serial UART [DEV-13712]. If you are using the Qwiic OpenLog for I^{2}C [DEV-15164], please refer to the Qwiic OpenLog Hookup Guide.

The OpenLog Data Logger is a simple-to-use, open-source solution for logging serial data from your projects. The OpenLog provides a simple serial interface to log data from a project to a microSD card.



SparkFun OpenLog
DEV-13712



SparkFun OpenLog with Headers

DEV-13955

no product found

Materials Required

In order to fully work through this tutorial, you will need the following parts. You may not need everything though depending on what you have. Add it to your cart, read through the guide, and adjust the cart as necessary.

OpenLog Hookup Guide SparkFun Wish List

Arduino Pro Mini 328 - 3.3V/8MHz
DEV-11114
It's blue! It's thin! It's the Arduino Pro Mini! SparkFun's minimal design approach to Arduino. This is a 3.3V Arduino …

SparkFun FTDI Basic Breakout - 3.3V
DEV-09873
This is the newest revision of our [FTDI Basic](http://www.sparkfun.com/commerce/product_info.php?products_id=…

SparkFun Cerberus USB Cable - 6ft
CAB-12016
You've got the wrong USB cable. It doesn't matter which one you have, it's the wrong one. But what if you could ha…

SparkFun OpenLog
DEV-13712
The SparkFun OpenLog is an open source data logger that works over a simple serial connection and supports mi…

microSD Card with Adapter - 16GB (Class 10)
COM-13833
This is a class 10 16GB microSD memory card, perfect for housing operating systems for single board computers a…

microSD USB Reader
COM-13004
This is an awesome little microSD USB reader. Just slide your microSD card into the inside of the USB connector, t…

Female Headers
PRT-00115
Single row of 40-holes, female header. Can be cut to size with a pair of wire-cutters. Standard .1" spacing. We use …

Jumper Wires Premium 6" M/M Pack of 10
PRT-08431
This is a SparkFun exclusive! These are 155mm long jumpers with male connectors on both ends. Use these to ju…

Break Away Male Headers - Right Angle
PRT-00553
A row of right angle male headers - break to fit. 40 pins that can be cut to any size. Used with custom PCBs or gen…

Recommended Reading

If you are not familiar or comfortable with the following concepts, we recommend reading through these before continuing on with the OpenLog Hookup Guide.

How to Solder: Through-Hole Soldering
This tutorial covers everything you need to know about through-hole soldering.

Serial Communication
Asynchronous serial communication concepts: packets, signal levels, baud rates, UARTs and more!

Serial Peripheral Interface (SPI)
SPI is commonly used to connect microcontrollers to peripherals such as sensors, shift registers, and SD cards.

Serial Terminal Basics
This tutorial will show you how to communicate with your serial devices using a variety of terminal emulator applications.

Hardware Overview

Power

The OpenLog runs at the following settings:

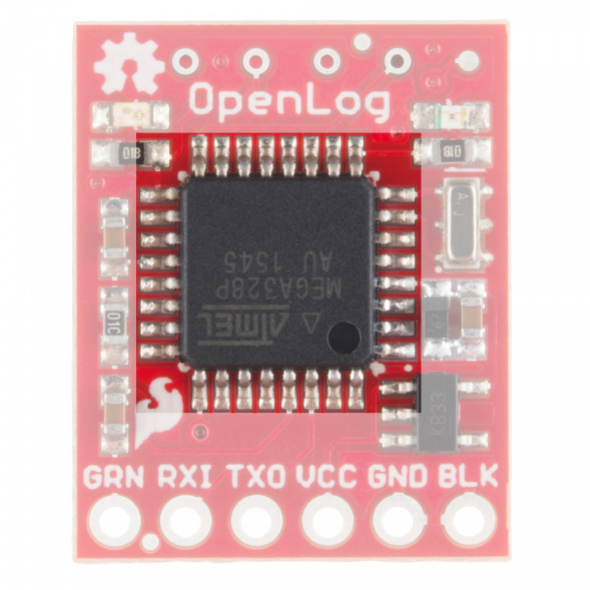
OpenLog Power Ratings

VCC Input	**3.3V-12V (Recommended 3.3V-5V)**
RXI Input	2.0V-3.8V
TXO Output	3.3V
Idle Current Draw	~2mA-5mA (w/out microSD card), ~5mA-6mA (w/ microSD card)
Active Writing Current Draw	**~20-23mA (w/ microSD card)**

The OpenLog's current draw is about **20mA to 23mA** when writing to a microSD. Depending on the size of the microSD card and its manufacturer, the active current draw can vary when the OpenLog is writing to the memory card. Increasing the baud rate will also pull more current.

Microcontroller

The OpenLog runs off of an onboard ATmega328, running at 16MHz thanks to the onboard crystal. The ATmega328 has the Optiboot bootloader loaded on it, which allows the OpenLog to be compatible with the "**Arduino Uno**" board setting in the Arduino IDE.

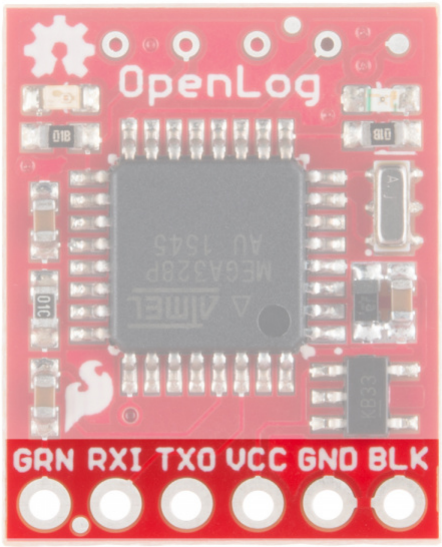


The brain of the OpenLog.

Interface

Serial UART

The primary interface with the OpenLog is the FTDI header on the board edge. This header is designed to plug directly into an Arduino Pro or Pro Mini, which allows the microcontroller to send the data over a serial connection to the OpenLog.



Serial Connection Header.

Warning! Because of the pin ordering that makes it compatible with the Arduinos, it **cannot** plug directly into an FTDI breakout board.

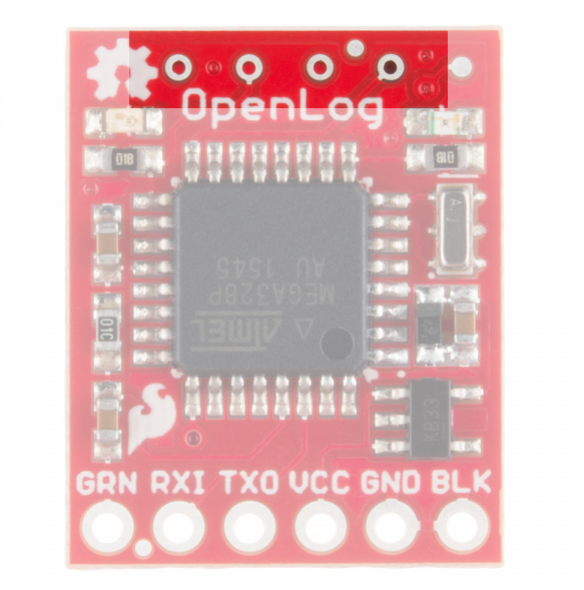
SparkFun FTDI Basic Breakout - 3.3V
DEV-09873

SparkFun USB Mini-B Cable - 6 Foot
CAB-11301

For more information, make sure to check out the next section on the Hardware Hookup.

SPI

There are also four SPI test points broken out on the opposite end of the board. You can use these to reprogram the bootloader on the ATmega328.



SPI Test Points.

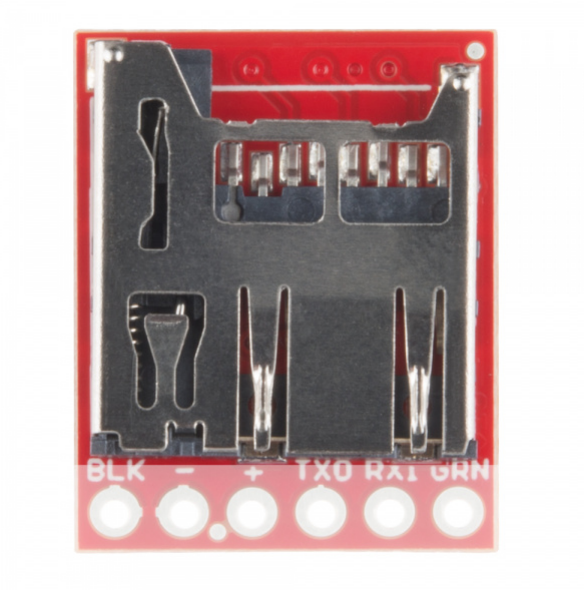
The latest OpenLog (DEV-13712) breaks out these pins on smaller plated through holes. If you need to use an ISP to reprogram or upload a new bootloader to the OpenLog, you can use pogo pins to connect to these test points.

The final interface for communicating with the OpenLog is the microSD card itself. To communicate, the microSD card requires SPI pins. Not only is this where the data is stored by the OpenLog, but you can also update the OpenLog's configuration via the `config.txt` file on the microSD card.

microSD Card

All data logged by the OpenLog is stored on the microSD card. The OpenLog works with microSD cards that involve the following features:

- 64MB to 32GB
- FAT16 or FAT32

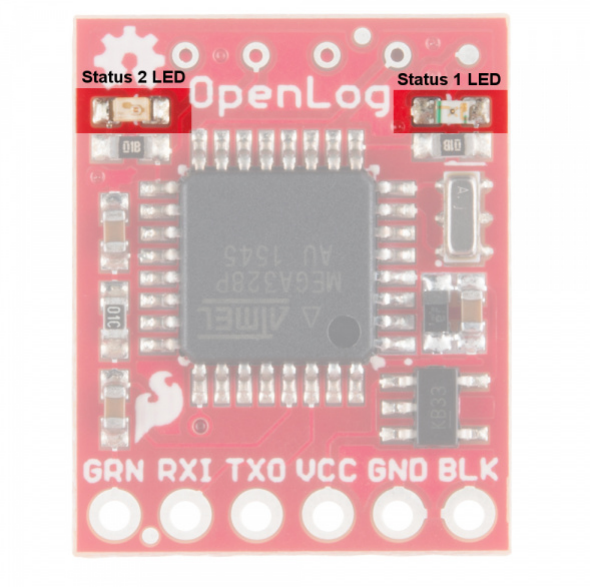


microSD Slot on the bottom of the OpenLog.

Status LED

There are two status LEDs on the OpenLog to help you with troubleshooting.

- **STAT1** - This blue indicator LED is attached to Arduino **D5** (ATmega328 PD5) and toggles on/off when a new character is received. This LED blinks when Serial communication is functioning.
- **STAT2** - This green LED is connected to Arduino **D13** (SPI Serial Clock Line/ ATmega328 PB5). This LED only blinks when the SPI interface is active. You will see it flash when the OpenLog records 512 bytes to the microSD card.



Status LEDs on the OpenLog.

Hardware Hookup

There are two main methods for connecting your OpenLog to a circuit. You will need some headers or wires to connect. Make sure that you solder to the board for a secure connection.

Basic Serial Connection

Tip: If you have a female header the OpenLog and female header on the FTDI you will need M/F jumper wires to connect.

Female Headers
PRT-00115

Jumper Wires Premium 6" M/M Pack of 10
PRT-08431

This hardware connection is designed for interfacing with an OpenLog if you need to reprogram the board, or log data over a basic serial connection.

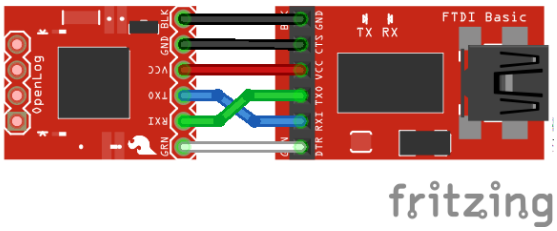
Make the following connections:

OpenLog → 3.3V FTDI Basic Breakout

- GND → GND
- GND → GND
- VCC → 3.3V
- TXO → RXI
- RXI → TXO
- DTR → DTR

Notice that it is **not** a direct connection between the FTDI and OpenLog - you must switch the TXO and RXI pin connections.

Your connections should look like the following:



Once you have the connections between the OpenLog and the FTDI Basic, plug your FTDI board into a USB cable and into your computer.

Open up a serial terminal, connect to the COM port of your FTDI Basic, and go to town!

Project Hardware Connection

Tip: If you have the female headers soldered on the OpenLog, you can solder male headers to the Arduino Pro Mini to plug the boards together without the need for wires.

Break Away Headers - Straight
PRT-00116

Break Away Male Headers - Right Angle
PRT-00553

While interfacing with the OpenLog over a serial connection is important for reprogramming or debugging, the place where OpenLog shines is in an embedded project. This general circuit is how we recommend you hook up your OpenLog to a microcontroller (in this case, an Arduino Pro Mini) that will write serial data out to the OpenLog.

First you will need to upload the code to your Pro Mini you intend to run. Please check out the Arduino Sketches for some example code that you can use.

Note: If you are unsure how to program your Pro Mini, please check out our tutorial here.

Using the Arduino Pro Mini 3.3V
This tutorial is your guide to all things Arduino Pro Mini. It explains what it is, what it's not, and how to get started using it.

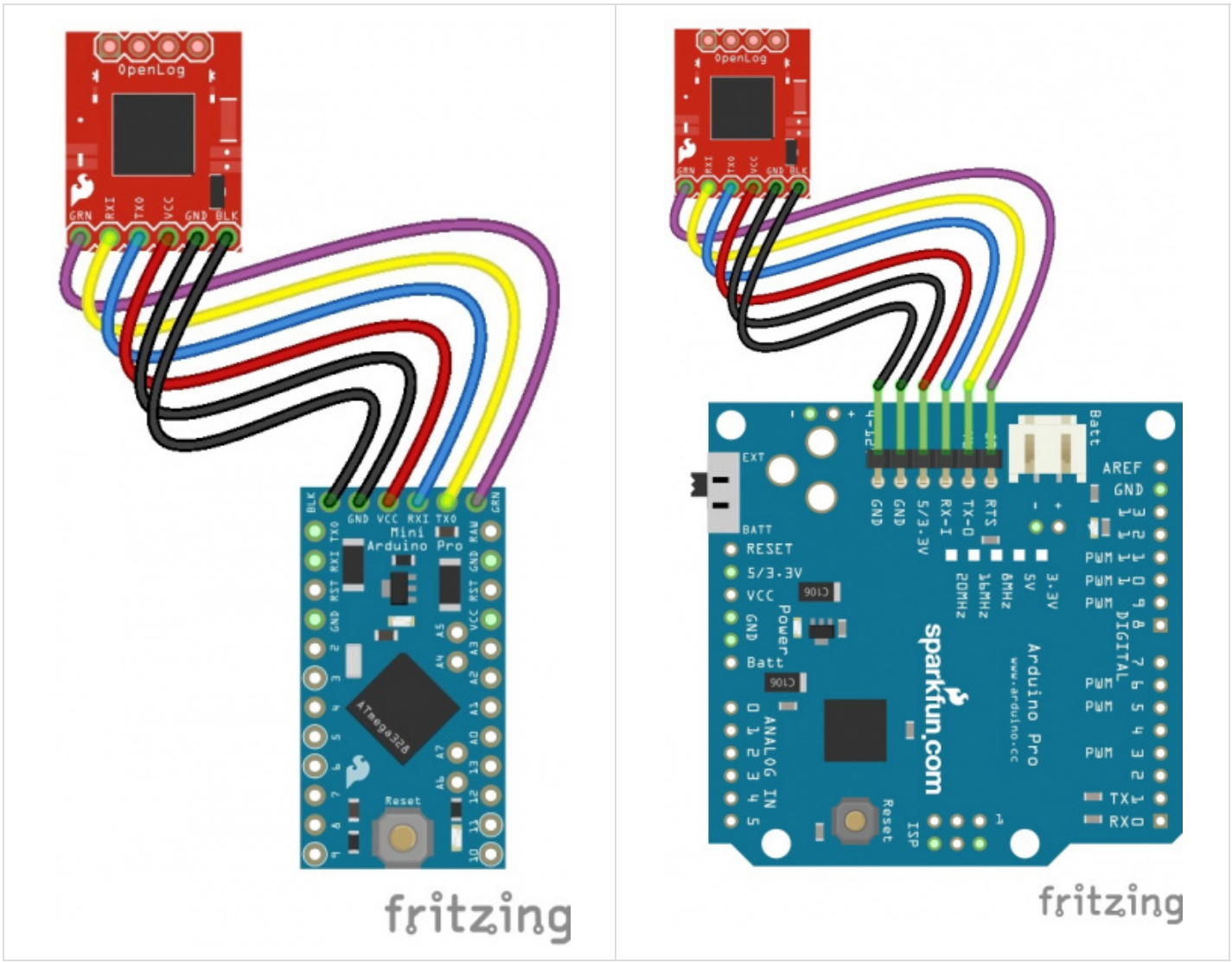
Once you have programmed your Pro Mini, you can remove the FTDI board, and replace it with the OpenLog. **Make sure** to connect the pins labeled `BLK` on both the Pro Mini and OpenLog (the pins labeled `GRN` on both will also match up if done correctly).

If you cannot plug the OpenLog directly into the Pro Mini (due to mismatched headers or other boards in the way), you can use jumper wires and make the following connections.

OpenLog → Arduino Pro/Arduino Pro Mini

- GND → GND
- GND → GND
- VCC → VCC
- TXO → RXI
- RXI → TXO
- DTR → DTR

Once you're finished, your connections should look like the following with the Arduino Pro Mini and Arduino Pro. The Fritzing diagram shows the OpenLogs with the headers mirrored. If you flip the microSD socket relative to the Arduino's top view, they should match the programming header like an FTDI.



Note that the connection is a straight shot with the OpenLog "upside-down" (with the microSD facing up).

⚡**Note:** Since Vcc and GND between the OpenLog and Arduino are being occupied by the headers, you'll need to connect to power to the other pins available on the Arduino. Otherwise, you could solder wires to the exposed power pins on either board.

Power up your system, and you are ready to start logging!

Arduino Sketches

There are six different examples sketches included that you can use on the Arduino when connected to an OpenLog.

- **OpenLog_Benchmarking** -- This example is used to test OpenLog. This sends very large amounts of data at 115200bps over multiple files.
- **OpenLog_CommandTest** -- This example shows how to create and append a file via command line control through the Arduino.
- **OpenLog_ReadExample** -- This example runs through how to control the OpenLog via command line.
- **OpenLog_ReadExample_LargeFile** -- Example of how to open a large stored file on OpenLog and report it over a local bluetooth connection.

- **OpenLog_Test_Sketch** -- Used to test OpenLog with lots of serial data.
- **OpenLog_Test_Sketch_Binary** -- Used to test OpenLog with binary data and escape characters.

Firmware

The OpenLog has two primary pieces of software on board: the bootloader and the firmware.

Arduino Bootloader

> **Note:** If you are using an OpenLog that was purchased prior to March 2012, the onboard bootloader is compatible with the "**Arduino Pro or Pro Mini 5V/16MHz w/ ATmega328**" setting in the Arduino IDE.

As mentioned previously, the OpenLog has the Optiboot serial bootloader on board. You can treat the OpenLog just like an **Arduino Uno** when uploading example code or new firmware to the board.

If you end up bricking your OpenLog and need to reinstall the bootloader, you will also want to upload Optiboot onto the board. Please check out our tutorial on installing an Arduino Bootloader for more information.

Compiling and Loading Firmware onto the OpenLog

> **Note:** If this is your first time using Arduino, please review our tutorial on installing the Arduino IDE. If you have not previously installed an Arduino library, please check out our installation guide to manually install the libraries.

If for any reason you need to update or reinstall the firmware on your OpenLog, the following process will get your board up and running.

First, please download the **Arduino IDE v1.6.5**. Other versions of the IDE **may** work to compile the OpenLog firmware, but we have verified this as a known good version.

Next, download the OpenLog firmware and required libraries bundle.

DOWNLOAD OPENLOG FIRMWARE BUNDLE (ZIP)

Once you have the libraries and firmware downloaded, install the libraries into Arduino. If you are unsure how to manually install the libraries in the IDE, please check out our tutorial: Installing an Arduino Library: Manually Installing a Library.

> **Note:** We are using modified versions of the SdFat and SerialPort libraries in order to arbitrarily declare how big the TX and RX buffers should be. The OpenLog requires the TX buffer to be very small (0) and the RX buffer needs to be as large as possible. Using these two modified libraries together allows increased performance of the OpenLog.

> **Looking for the Latest Versions?** If you would prefer the most up-to-date versions of the libraries and firmware, you can download them directly from the GitHub repositories linked below. The SdFatLib and Serial Port libraries are not visible in the Arduino board manager so you will need to manually install the library.

- GitHub: OpenLog > Firmware > OpenLog_Firmware
- Bill Greiman's Arduino Libraries
    - SdFatLib-beta
    - SerialPort

Next, to take advantage of the modified libraries, modify the *SerialPort.h* file found in **\Arduino\Libraries\SerialPort** directory. Change `BUFFERED_TX` to `0` and `ENABLE_RX_ERROR_CHECKING` to `0`. Save the file, and open up the Arduino IDE.

If you haven't yet, connect your OpenLog to the computer via an FTDI board. Please double check the example circuit if you are not sure how to do this properly.

Open the OpenLog sketch you would like to upload under **Tools>Board** menu, select the *"Arduino/Genuino Uno"*, and select the proper *COM port* for your FTDI board under **Tools>Port**.

Upload the code.

That's it! Your OpenLog is now programmed with new firmware. You can now open up a serial monitor and interact with the OpenLog. On power up, you will see either `12>` or `12<`. `1` indicates the serial connection is established, `2` indicates the SD card has successfully initialized, `<` indicates OpenLog is ready to log any received serial data and `>` indicates OpenLog is ready to receive commands.

OpenLog Firmware Sketches

There are three included sketches you can use on the OpenLog, depending on your particular application.

- **OpenLog** - This firmware ships by **default** on the OpenLog. Sending the `?` command will show the firmware version loaded onto a unit.
- **OpenLog_Light** - This version of the sketch removes the menu and command mode, allowing the receive buffer to be increased. This is a good option for high-speed logging.
- **OpenLog_Minimal** - The baud rate must be set in code and uploaded. This sketch is recommended for experienced users but is also the best option for the highest speed logging.

Command Set

You can interface with the OpenLog via a serial terminal. The following commands will help you read, write, and delete files, as well as change the settings of the OpenLog. You will need to be in `Command Mode` in order to use the following settings.

While the OpenLog is in `Command Mode`, STAT1 will toggle on/off for every character received. The LED will stay on until the next character is received.

File Manipulation

- **`new File`** - Creates a new file named *File* in the current directory. Standard 8.3 filenames are supported. For example, "87654321.123" is acceptable, while "987654321.123" is not.
    - Example: `new file1.txt`
- **`append File`** - Append text to the end of *File*. Serial data is then read from the UART in a stream and adds it to the file. It is not echoed over the serial terminal. If *File* does not exist when this function is called, the file will be created.
    - Example: `append newfile.csv`

- **`write File OFFSET`** - Write text to *File* from the location *OFFSET* within the file. The text is read from the UART, line by line and echoed back. To exit this state, send an empty line.
    - Example: `write logs.txt 516`
- **`rm File`** - Deletes *File* from the current directory. Wildcards are supported.
    - Example: `rm README.txt`
- **`size File`** - Output size of *File* in bytes.
    - Example: `size Log112.csv`
    - Output: `11`
- **`read File + START+ LENGTH TYPE`** - Output the content of *File* starting from *START* and going for *LENGTH*. If *START* is omitted, the entire file is reported. If *LENGTH* is omitted, the entire contents from the starting point is reported. If *TYPE* is omitted, the OpenLog will default to reporting in ASCII. There are three output *TYPEs*:
    - ASCII = 1
    - HEX = 2
    - RAW = 3

  You may leave off some trailing arguments. Check the following examples.

  Basic read + omitted flags:
    - Example: `read LOG00004.txt`
    - Output: `Accelerometer X=12 Y=215 Z=317`

  Read from start 0 with length of 5:
    - Example: `read LOG00004.txt 0 5`
    - Output: `Accel`

  Read from position 1 with a length of 5 in HEX:
    - Example: `read LOG00004.txt 1 5 2`
    - Output: `63 63 65 6C`

  Read from position 0 with a length of 50 in RAW:
    - Example: `read LOG00137.txt 0 50 3`
    - Output: `André-- -þ Extended Character Test`
- **`cat File`** - Write the content of a file in hex to the serial monitor for viewing. This is sometimes helpful to see that a file is recording correctly without having to pull the SD card and view the file on a computer.
    - Example: `cat LOG00004.txt`
    - Output: `00000000: 41 63 65 6c 3a 20 31`

Directory Manipulation

- **`ls`** - Lists all contents of the current directory. Wildcards are supported.
    - Example: `ls`
    - Output: `\src`
- **`md Subdirectory`** - Create *Subdirectory* in the current directory.
    - Example: `md Example_Sketches`
- **`cd Subdirectory`** - Change to *Subdirectory*.
    - Example: `cd Hello_World`

- **`cd ..`** - Change to a lower directory in the tree. **Note that there is a space between 'cd' and '..'.** This allows the string parser to see the `cd` command.
    - Example: `cd ..`
- **`rm Subdirectory`** - Deletes *Subdirectory*. The directory must be empty for this command to work.
    - Example: `rm temps`
- **`rm -rf Directory`** - Deletes *Directory* and any files contained within it.
    - Example: `rm -rf Libraries`

Low Level Function Commands

- **`?`** - This command will pull up a list of available commands on the OpenLog.
- **`disk`** - Show card manufacturer ID, serial number, manufacture date and card size. Example output is:

```
Card type: SD2
Manufacturer ID: 3
OEM ID: SD
Product: SU01G
Version: 8.0
Serial number: 39723042
Manufacturing date: 1/2010
Card Size: 965120 KB
```

- **`init`** - Reinitialize the system and reopen the SD card. This is helpful if the SD card stops responding.
- **`sync`** - Synchronizes the current contents of the buffer to the SD card. This command is useful if you have less than 512 characters in the buffer and want to record those on the SD card.
- **`reset`** - Jumps OpenLog to location zero, reruns bootloader and then init code. This command is helpful if you need to edit the config file, reset the OpenLog and start using the new configuration. Power cycling is still the preferred method for resetting the board, but this option is available.

System Settings

These settings can be manually updated, or edited in the `config.txt` file.

- **`echo STATE`** - Changes the state of the system echo, and is stored in system memory. *STATE* can either be `on` or `off`. While `on`, the OpenLog will echo received serial data on the command prompt. While `off`, the system does not read back received characters.

> **Note:** During normal logging, echo will be turned off. The system resource demands for echoing the received data is too high during logging.

- **`verbose STATE`** - Changes the state of verbose error reporting. *STATE* can either be `on` or `off`. This command is stored in memory. By turning off verbose errors, OpenLog will respond with only a `!` if there is an error rather than `unknown command: COMMAND`. The `!` character is easier for embedded systems to parse than the full error. If you are using a terminal, leaving `verbose on` will allow you to see full error messages.

- `baud` - This command will open up a system menu allowing the user to enter a baud rate. Any baud rate between 300bps and 1Mbps is supported. The baud rate selection is immediate, and the OpenLog requires a power cycle for the settings to take effect. The baud rate is stored to EEPROM and is loaded every time OpenLog powers up. The default is `9600 8N1`.

Remember: If you get the board stuck in an unknown baud rate, you can tie RX to GND and power up OpenLog. The LEDs will blink back and forth for 2 seconds and will then blink in unison. Power down the OpenLog, and remove the jumper. OpenLog is now reset to 9600bps with an escape character of `CTRL-Z` pressed three consecutive times. This feature can be overridden by setting the Emergency Override bit to 1. See config.txt for more info.

- `set` - This command opens a system menu to select the boot up mode. These settings will occur at the next power-on and are stored in non-volatile EEPROM.
    - **New File Logging** - This mode creates a new file each time OpenLog powers up. OpenLog will transmit `1` (UART is alive), `2` (SD card is initialized), then `<` (OpenLog is ready to receive data). All data will be recorded to a `LOG#####.txt`. The ##### number increases every time OpenLog powers up (the max is 65533 logs). The number is stored in EEPROM and can be reset from the `set` menu. All received characters are not echoed. You can exit this mode and enter command mode by sending `CTRL+z` (ASCII 26). All buffered data will be stored.

Note: If too many logs have been created, OpenLog will output error **Too many logs**, exit this mode, and drop to Command Prompt. The serial output will look like `12!Too many logs!`.

    - **Append File Logging** - Also known as sequential mode, this mode creates a file called `SEQLOG.txt` if it is not already there, and appends any received data to the file. OpenLog will transmit `12<` at which time OpenLog is ready to receive data. Characters are not echoed. You can exit this mode and enter command mode by sending `CTRL+z` (ASCII 26). All buffered data will be stored.
    - **Command Prompt** - OpenLog will transmit `12>` at which time the system is ready to receive commands. Note that the `>` sign indicates OpenLog is ready to receive commands, **not** data. You can create files and append data to files, but this requires some serial parsing (for error checking), so we do not set this mode by default.
    - **Reset New File Number** - This mode will reset the log file number to `LOG000.txt`. This is helpful if you have recently cleared out a microSD card and want the log file numbers to start over again.
    - **New Escape Character** - This option allows the user to enter a character such as `CTRL+z` or `$`, and set this as the new escape character. This setting is reset to `CTRL+z` during an emergency reset.
    - **Number of Escape Characters** - This option allows the user to enter a character (such as 1, 3, or 17), updating the new number of escape characters needed to drop to command mode. For example, entering 8 will require the user to hit `CTRL+z` eight times to get to command mode. This setting is reset to 3 during an emergency reset.

Escape Characters Explanation: The reason OpenLog requires `CTRL+z` hit 3 times to enter command mode is to prevent the board accidentally being reset during upload of new code from the Arduino IDE. There is the chance that the board would see the `CTRL+z` character coming through during bootloading (an issue we saw in the early versions of the OpenLog firmware), so this aims to

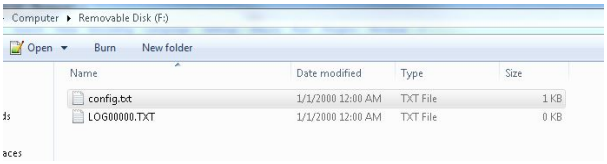
prevent that. If you ever suspect your board has been bricked due to this, you can always do an emergency reset by holding the RX pin to ground during power up.

Configuration File

If you would rather not use the serial terminal for modifying the settings on your OpenLog, you can also update the settings by modifying the **CONFIG.TXT** file.

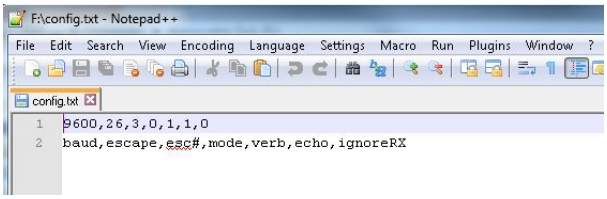
Note: This feature only functions on firmware verison 1.6 or newer. If you have bought an OpenLog after 2012, you will be running firmware version 1.6+

To do this, you will need a microSD card reader and a text editor. Open up the **config.txt** file (the capitalization of the file name does not matter), and configure away! If you have never powered up your OpenLog with the SD card before, you can also manually create the file. If you have powered up the OpenLog with the microSD card inserted previously, you should see something like the following when you read the microSD card.



The OpenLog creates a config.txt and LOG0000.txt file on first power up.

The default configuration file has one line of settings and one line of definitions.



Default configuration file written by the OpenLog.

Note that these are regular visible characters (there are no non-visible or binary values), and each value is separated by a comma.

The settings are defined as follows:

- **`baud`** : The communication baud rate. 9600bps is default. Acceptable values that are compatible with the Arduino IDE are 2400, 4800, 9600, 19200, 38400, 57600, and 115200. You can use other baud rates, but you will be unable to communicate with the OpenLog through the Arduino IDE serial monitor.
- **`escape`** : The ASCII value (in decimal format) of the escape character. 26 is `CTRL+z` and is default. 36 is `$` and is a commonly used escape character.
- **`esc#`** : The number of escape characters required. By default, it is three, so you must hit the escape character three times to drop to command mode. Acceptable values are from 0 to 254. Setting this value to `0` will disable escape character checking completely.
- **`mode`** : System mode. OpenLog starts in New Log mode(`0`) by default. Acceptable values are `0` =New Log, `1` = Sequential Log, `2` = Command Mode.

- **`verb`** : Verbose mode. Extended (verbose) error messages are turned on by default. Setting this to `1` turns on verbose error messages (such as `unknown command: remove !`). Setting this to `0` turns off verbose errors but will respond with a `!` if there is an error. Turning off verbose mode is handy if you are trying to handle errors from an embedded system.
- **`echo`** : Echo mode. While in command mode, characters are echoed by default. Setting this to `0` turns off character echo. Turning this off is handy if handling errors and you don't want sent commands being echoed back to the OpenLog.
- **`ignoreRX`** : Emergency Override. Normally, OpenLog will emergency reset when the RX pin is pulled low during power up. Setting this to `1` will disable the checking of the RX pin during power up. This can be helpful for systems that will hold the RX line low for various reasons. If Emergency Override is disabled, you **will not** be able to force the unit back to 9600bps, and the configuration file will be the only way to modify the baud rate.

How OpenLog Modifies the Config File

There are five different situations for the OpenLog to modify the `config.txt` file.

- **Config file found**: During power up, OpenLog will look for a `config.txt` file. If the file is found, OpenLog will use the included settings and overwrite any previously stored system settings.
- **No config file found**: If OpenLog cannot find the `config.txt` file then OpenLog will create `config.txt` and record the currently stored system settings to it. This means if you insert a newly formatted microSD card, your system will maintain its current settings.
- **Corrupt config file found**: OpenLog will erase the corrupted `config.txt` file, and will rewrite both the internal EEPROM settings and the `config.txt` settings file to the known-good state of `9600,26,3,0,1,1,0`.
- **Illegal values in config file**: If the OpenLog discovers any settings containing illegal values, OpenLog will overwrite the corrupt values in `config.txt` file with the currently stored EEPROM system settings.
- **Changes through command prompt**: If the system settings are changed through the command prompt (either over a serial connection or via microcontroller serial commands) those changes will be recorded both to the system EEPROM and to the `config.txt` file.
- **Emergency Reset**: If the OpenLog is power cycled with a jumper between RX and GND, and the Emergency Override bit is set to `0` (allowing emergency reset), OpenLog will rewrite both the internal EEPROM settings and the `config.txt` settings file to the known-good state of `9600,26,3,0,1,1,0`.

Troubleshooting

There are several different options to check if you are having issues connecting over the serial monitor, having issues with dropped characters in logs, or fighting a bricked OpenLog.

Check STAT1 LED Behavior

STAT1 LED shows different behavior for two different common errors.

- 3 Blinks: The microSD card failed to initialize. You may need to format the card with FAT/FAT16 on a computer.
- 5 Blinks: OpenLog has changed to a new baud rate and needs to be power cycled.

Double Check Subdirectory Structure

If you are using the default OpenLog.ino example, OpenLog will only support two subdirectories. You will need to change `FOLDER_TRACK_DEPTH` from `2` to the number of subdirectories you need to support. Once you've done this, recompile the code up, and upload the modified firmware.

Verify the Number of Files in the Root Directory

OpenLog will only support up to 65,534 log files in the root directory. We recommend reformatting your microSD card to improve logging speed.

Verify the Size of your Modified Firmware

If you are writing a custom sketch for the OpenLog, verify that your sketch is not larger than 32,256. If so, it will cut into the upper 500 bytes of Flash memory, which is used by the Optiboot serial bootloader.

Double Check File Names

All file names should be alpha-numeric. `MyLOG1.txt` is ok, but `Hi !e _.txt` may not work.

Use 9600 Baud

OpenLog runs off of the ATmega328 and has a limited amount of RAM(2048 bytes). When you send serial characters to OpenLog, these characters get buffered. The SD Group Simplified Specification allows an SD card to take up to 250ms (section 4.6.2.2 Write) to record a data block to flash memory.

At 9600bps, that's 960 bytes (10 bits per byte) per second. That is 1.04ms per byte. OpenLog currently uses a 512 byte receive buffer so it can buffer around 50ms of characters. This allows OpenLog to successfully receive all characters coming at 9600bps. As you increase the baud rate, the buffer will last for less time.

OpenLog Buffer Overrun Time

Baud Rate	Time per byte	Time Until Buffer is Overrun
9600bps	1.04ms	532ms
57600bps	0.174ms	88ms
115200bps	0.087ms	44ms

Many SD cards have a faster record time than 250ms. This can be affected by the 'class' of the card and how much data is already stored on the card. The solution is to use a lower baud rate or increase the amount of time between the characters sent at the higher baud rate.

Format your MicroSD Card

Remember to use a card with few or no files on it. A microSD card with 3.1GB worth of ZIP files or MP3s has a slower response time than an empty card.

If you did not format your microSD card on a Windows OS, reformat the microSD card and create a DOS filesystem on the SD card.

Swap MicroSD Cards

There are many different types of card manufacturers, relabeled cards, card sizes, and card classes, and they may not all work properly. We typically use an 8GB class 4 microSD card, which works well at 9600bps. If you need higher baud rates, or larger storage space, you may want to try class 6 or above cards.

Add Delays Between Character Writes

By adding a small delay between Serial.print() statements, you can give OpenLog a chance to record its current buffer.

For example:

```
Serial.begin(115200);
for(int i = 1 ; i < 10 ; i++) {
    Serial.print(i, DEC);
    Serial.println(":abcdefghijklmnopqrstuvwxyz-!#");
}
```

may not log properly, as there are a lot of characters being sent right next to each other. Inserting a small delay of 15ms between large character writes will help OpenLog record without dropping characters.

```
Serial.begin(115200);
for(int i = 1 ; i < 10 ; i++) {
    Serial.print(i, DEC);
    Serial.println(":abcdefghijklmnopqrstuvwxyz-!#");
    delay(15);
}
```

Add Arduino Serial Monitor Compatibility

If you are attempting to use the OpenLog with the built-in serial library or the SoftwareSerial library, you may notice issues with command mode. `Serial.println()` sends both newline AND carriage return. There are two alternative commands to overcome this.

The first is to use the `\r` command (ASCII carriage return):

```
Serial.print("TEXT\r");
```

Alternatively, you can send the value 13 (decimal carriage return):

```
Serial.print("TEXT");
Serial.write(13);
```

Emergency Reset

Remember, if you need to reset the OpenLog to a default state, you can reset the board by tying the RX pin to GND, powering up the OpenLog, waiting until the LEDs begin to blink in unison, and then powering down the OpenLog and removing the jumper.

If you have changed the Emergency Override bit to `1`, you will need to modify the configuration file, as the Emergency Reset **will not** work.

Check with the Community

If you are still having issues with your OpenLog, please check out the current and closed issues on our GitHub repository here. There is a large community working with the OpenLog, so chances are that someone has found a fix for the problem you are seeing.

Resources and Going Further

Now that you've successfully logged data with your OpenLog, you can set up remote projects and monitor all the possible data coming. Consider creating your own Citizen Science project, or even a pet tracker to see what Fluffy does when out and about!

Check out these additional resources for troubleshooting, help, or inspiration for your next project.

- OpenLog GitHub
- Illumitune Project
- LilyPad Light Sensor Hookup
- BadgerHack: Soil Sensor Add-On
- Getting Started with OBD-II
- Vernier Photogate

Need some more inspiration? Check out some of these related tutorials:

Photon Remote Water Level Sensor

Learn how to build a remote water level sensor for a water storage tank and how to automate a pump based off the readings!

Blynk Board Project Guide

A series of Blynk projects you can set up on the Blynk Board without ever re-programming it.

Logging Data to Google Sheets with the Tessel 2

This project covers how to log data to Google Sheets two ways: using IFTTT with a web connection or a USB pen drive and "sneakernet" without.

Graph Sensor Data with Python and Matplotlib

Use matplotlib to create a real-time plot of temperature data collected from a TMP102 sensor connected to a Raspberry Pi.

If you have any tutorial feedback, please visit the comments or contact our technical support team at TechSupport@sparkfun.com.