public sparkfun / OpenLog

Command Set

Upon power up, in default 'NewLog' mode, OpenLog will output '**12<**' (note the less than sign) at which time you can start throwing characters at it. Those characters will be recorded to LOG#####.TXT. Press Ctrl+z (ASCII 26 decimal) three consecutive times will cause OpenLog to exit record mode and drop into command mode. Command mode is signified with a '>' (greater than) sign. You can then enter the following commands:

Remember, once you're in command mode the list of available commands can be brought up by typing '?'.

Basic Commands

new *file*: Creates a new file (named *file*) in the current directory. Only standard 8.3 file names are supported. "12345678.123" is acceptable. "123456789.123" is not.

- Ex: **new file1.txt**

append *file*: Appends text to the end of *file*. Serial data is then read in from the UART in a stream and is not echoed. This is the most common way to use the logger. To exit this state, send the command Ctrl+z (ASCII 26) and OpenLog will return to the '>' prompt. If the file does not exist it will be created.

- Ex: **append newfile.csv**

write *file* offset: Writes text to *file* starting 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.

- Ex: **write logs.f17 516**

rm *file*: Deletes *file* from the current directory. Wild cards are supported starting in version 1.2.

- Ex: **rm me.txt**
    - In v2.4 and above: You can remove directories and any files that may be contained in them.
    - Ex: **rm -rf mydirectory**

size *file*: Displays size of *file* in bytes.

- Ex: **size LOG00004.txt**
- Output: **11**

Directory Manipulation

ls: Shows the content of the current directory. Wild cards are supported starting in version 1.2.

- Ex: **ls**

md *directory*: Creates a sub directory in the current directory.

- Ex: **md july_14**

cd *directory*: Change to a sub directory

- Ex: **cd day_22**

cd ..: Change to lower directory in tree. This is a bit weird because I'm used to "cd..". There is a space in between **cd** and **..** to allow the string parser to see the **cd** command.

- Ex: **cd ..**

rm *directory*: Remove a sub directory

- Ex: **rm temps**
- Note that the directory must be empty for this command to work. To remove a directory with files in it, use the **rm -rf temps** command.

Viewing Files

read *file* start length TYPE: (New in v1.2) Outputs the contents of a file in visible form of a *file* starting from start and going to length. There are three TYPEs of output, ASCII = 1, HEX = 2, and RAW = 3. You may leave off any of the arguments.

- Ex: **read LOG00004.TXT 0 5**
- Output: **Accel**
- Ex: **read LOG00004.TXT**

- Output: **Accelerometer X=12 Y=215 Z=317**
- Ex: **read LOG00004.TXT 1 5 2**
- Output: **63 63 65 6C**
- Ex: **read LOG00137.TXT 0 50 3**
- Output: **André—**
  -þ Extended Character Test (note that printing option RAW will depend on how your terminal will handle the extended characters)

cat *file*: Writes the contents of a file in hex to the screen 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.

- Ex: **cat LOG00004.txt**
- Output:
  00000000: 41 63 63 65 6c 3a 20 31
  00000008: 32 32 0d

System Settings

Remember, on version 1.6 and above, system configuration can be done via the **CONFIG.TXT** file.

echo *<on/off>*: New in version 2.3. This command allows you to turn on or off echo. This command is not stored in memory and defaults to **echo on** at each power up. By turning echo off, OpenLog will not echo typed text on the command prompt. This is helpful if OpenLog is embedded into a system where the system does not read back the echoed characters. If you are playing with OpenLog via a terminal, you probably want to leave **echo on** so that you can see the text you are sending to OpenLog. *Note*: During an append you will never see the text you are typing. This is because it takes too much system resources to echo back the text that is being received.

verbose *<on/off>*: New in version 2.3. This command turns on or off verbose error reporting. This command is not stored in memory and defaults to **verbose on** at each power up. By turning off verbose errors, OpenLog will respond with only a '!' if there is an error rather that "unknown command: oodf" (for example). This '!' character is easier for embedded systems to parse than the full error. If you are playing with OpenLog via a terminal, you probably want to leave **verbose on** so that you can see full error messages.

baud: Brings up a system menu to select a baud rate. **9600 8N1 is the default**. You will be able to select either **2400**, **4800**, **9600**, **19200**, **57600**, and **115200** bit per second baud rates. Additional baud rates are achievable but these are the most common baud rates IMO, so that's all I've added. The baud rate selection is immediate and 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.

Remember, if you get OpenLog stuck into an unknown baud rate, there is a safety mechanism built-in. Tie the RX pin to ground and power up OpenLog. You should see the LEDs blink back and forth for 2 seconds, then blink in unison. Now power down OpenLog and remove the RX/GND jumper. OpenLog is now reset to **9600bps** with an escape character of 'ctrl+z' pressed three consecutive times.

set: Brings up 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:

1. **New File Logging** (default) : This mode creates a new file each time OpenLog powers up. OpenLog will transmit '**1**' (UART is alive) then '**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 you power up OpenLog. This 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 (65533 logs is max), OpenLog will output error '**!Too many logs:1!**', then exit this mode, then it will drop to Command Prompt. If you are parsing OpenLog output, look for '**12<**' before sending serial strings. If you see '**12!**Too many logs…' instead, you know there has been an error.
2. **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. 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.
3. **Command Prompt**: OpenLog will transmit '**12>**' at which time OpenLog is ready to receive *commands*. Note the *"greater than"* sign indicates OpenLog is now ready to receive commands, not data. You can easily create files and append data to files from the command prompt. This however requires a few more steps and some serial parsing (to check for errors) so we do not set this mode by default.
4. **Reset New File Number**: Selecting this option will reset the log file number to LOG000.txt. This is helpful if you've recently cleared out an SD card and want the log file numbers to start over again.
5. **New escape character** (new in v1.51): OpenLog will then prompt user to enter a character such as ctrl+z or '**$**'. This will be the new escape character. This setting is reset to ctrl+z during an emergency reset.
6. **Number of escape characters** (new in v1.51): OpenLog will then prompt user to enter a character such as 1 or 3 or 17 (that's just silly!). This will be the new number of escape characters needed to drop to command mode. For example, 3 will require the user to hit ctrl+z three times to get to command mode. This setting is reset to 3 during an emergency reset.

Escape characters explained:
Original OpenLog firmware (versions v1.5 and below) required you to press ctrl+z to enter command mode. What we found was that many users had OpenLog attached to the TX pin on an Arduino. When you hit the 'Upload' button in the Arduino IDE, serial characters start flowing to the Arduino board in rapid succession – the Arduino is reprogramming! The problem is that there is a strong chance that one of these reprogramming characters will be ctrl+z (ASCII 26) causing OpenLog (who is diligently logging all these crazy characters as it 'listens' to the TX pin) to drop to command mode. Now when you go to run your sketch, OpenLog is not responding? That's because it's sitting in command mode. Firmware v1.51 and above fixes this issue by setting the escape sequence to three ctrl+z characters. You have to hit ctrl+z *three* times to get it to drop to command mode. During a bootload of an Arduino, it is highly unlikely (but still possible) to see three such characters stream by. Both the type of escape character and the number of required escape characters are user configurable. '**$$$**' is a common configuration string for old AT systems. We decided to stick with ctrl+z because that's what we've been using since the beginning. **Remember:** if things get out of hand and you forget what escape character you've set, or how many you need to hit, you can always do an emergency reset by holding the RX pin to ground during a power up. This will cause OpenLog to default to 9600bps, with escape character of ctrl+z sent three times.

Note for Arduino users:
If you are using openlog with either the inbuilt serial library, or new software serial library, you may have noticed that command mode appears to be not working. This is because Serial.println appears to send both newline AND carriage return. To overcome this use the n.print("Your command\r") as shown, with a \r or separately send the value 13 by using Serial.write(13);

Low Level Commands

disk: Shows card manufacturer, status, file system capacity and free storage space.

init: Re-initializes the and re-opens the SD card. This is helpful if the SD card stops responding.

sync: Synchronize, or write 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 have them committed to the SD card.

reset: This will cause OpenLog to jump to location zero and re-run bootloader and then init code. This command is helpful if you need to edit the config file then have OpenLog reset and start using the new configuration. It's not a completely clean reset (watchdog would be better but does not currently work), so use with some caution.

Configuring OpenLog via a File

As of firmware version 1.6 and above, you no longer need to use a terminal to reconfigure OpenLog. Simply edit the **CONFIG.TXT** file (capitalization is important) and change *9600* to *57600* and OpenLog will operate at 57600bps the next time you power it up. This is pretty handy! Rather than hooking up a serial connection and opening up a terminal window, you can now edit many of the settings on OpenLog via an SD card reader and a text editor.

See the Config File page for more information.

OpenLog Wiki Pages:

- Command Set
- Datasheet
- Example Code
- Continuous Logging
- Config File
- Design Files
- Flashing Firmware
- Logger Comparison

Last edited by nseidle, 2 months ago

Markdown Cheat Sheet

Format Text

Headers

```
# This is an <h1> tag
## This is an <h2> tag
###### This is an <h6> tag
```

Text styles

```
*This text will be italic*
_This will also be italic_
**This text will be bold**
__This will also be bold__

*You **can** combine them*
```

Lists

Unordered

```
* Item 1
* Item 2
  * Item 2a
  * Item 2b
```

Ordered

```
1. Item 1
2. Item 2
3. Item 3
   * Item 3a
   * Item 3b
```

Miscellaneous

Images

```
![GitHub Logo](/images/logo.png)
Format: ![Alt Text](url)
```

Links

```
http://github.com - automatic!
[GitHub](http://github.com)
```

Blockquotes

```
As Kanye West said:

> We're living the future so
> the present is our past.
```

Code Examples in Markdown

Syntax highlighting with GFM

~~~
```javascript
function fancyAlert(arg) {
  if(arg) {
    $.facebox({div:'#foo'})
  }
}
```
~~~

Or, indent your code 4 spaces

```
Here is a Python code example
without syntax highlighting:

    def foo:
      if not bar:
        return true
```

Inline code for comments

```
I think you should use an
`<addr>` element here instead.
```

Something went wrong with that request. Please try again.