ALog

v0.3.2

Generated by Doxygen 1.8.13

Contents

1 ALog **1**

2 Class Documentation **55**

2.1 ALog Class Reference 55

2.1.1 Constructor & Destructor Documentation 56

2.1.1.1 ALog() 56

2.1.2 Member Function Documentation 57

2.1.2.1 _sensor_function_template() 57

2.1.2.2 analogReadOversample() 57

2.1.2.3 Anemometer_reed_switch() 58

2.1.2.4 Barometer_BMP180() 59

2.1.2.5 Decagon5TE() 59

2.1.2.6 DecagonGS1() 60

2.1.2.7 endAnalog() 60

2.1.2.8 endLogging() 60

2.1.2.9 get_3V3_measured_voltage() 61

2.1.2.10 get_5V_measured_voltage() 61

2.1.2.11 get_serial_number() 61

2.1.2.12 get_use_sleep_mode() 61

2.1.2.13 goToSleep_if_needed() 61

2.1.2.14 HackHD() 61

2.1.2.15 HM1500LF_humidity_with_external_temperature() . . . 63
2.1.2.16 Honeywell_HSC_analog() . . . 63
2.1.2.17 HTM2500LF_humidity_temperature() . . . 64
2.1.2.18 Inclinometer_SCA100T_D02_analog_Tcorr() . . . 65
2.1.2.19 initialize() . . . 66
2.1.2.20 linearPotentiometer() . . . 67
2.1.2.21 maxbotixHRXL_WR_analog() . . . 68
2.1.2.22 maxbotixHRXL_WR_Serial() . . . 69
2.1.2.23 Pyranometer() . . . 69
2.1.2.24 readPin() . . . 70
2.1.2.25 readPinOversample() . . . 70
2.1.2.26 sensorPowerOff() . . . 72
2.1.2.27 sensorPowerOn() . . . 72
2.1.2.28 set_LEDpin() . . . 72
2.1.2.29 set_RTCpowerPin() . . . 73
2.1.2.30 set_SDpowerPin() . . . 73
2.1.2.31 set_SensorPowerPin() . . . 73
2.1.2.32 setupLogger() . . . 74
2.1.2.33 sleep() . . . 74
2.1.2.34 startAnalog() . . . 74
2.1.2.35 startLogging() . . . 74
2.1.2.36 thermistorB() . . . 74
2.1.2.37 ultrasonicMB_analog_1cm() . . . 75
2.1.2.38 vdivR() . . . 76
2.1.2.39 Wind_Vane_Inspeed() . . . 77

3 File Documentation **79**

3.1 src/ALog.cpp File Reference . . . 79
3.1.1 Detailed Description . . . 80
3.1.2 Function Documentation . . . 81
3.1.2.1 read_Aref() . . . 81
3.1.2.2 save_Aref() . . . 81
3.2 src/ALog.h File Reference . . . 82
3.2.1 Detailed Description . . . 83

Index **85**

Chapter 1

ALog

The **ALog** library is a toolkit for open-source data logging designed for the Arduino-based ALog (http://northernwidget.com/alog/), but that will also work with any standard Arduino-Uno or -Mega-based system that is outfitted with a SD card and a DS3231 real-time clock.

This library is optimized to:

1. handle all of the basic file, system, and power management behind-the-scenes, including power consumption reduction to minimal levels through the use of the sleep functions

2. include and expose sensor functions as single-line calls, with a template for the addition of new sensors.

If you use the ALog library (and/or data logger) in a publication, please cite:

∗∗Wickert, A. D. (2014), [The ALog: Inexpensive, Open-Source, Automated Data Collection in the Field](http://onlinelibrary.wiley.com/wol1/doi/10.1890/0012-9623-95.2.68/full), *Bull. Ecol. Soc. Am.*, *95*(2), 68–78, <doi:10.1890/0012-9623-95.2.68.∗∗>

In addition to the README.md at https://github.com/NorthernWidget/ALog, documentation is available as a combination of the information here and an index of logger functions in both [HTML] and [PDF] format.

Quick reference

\

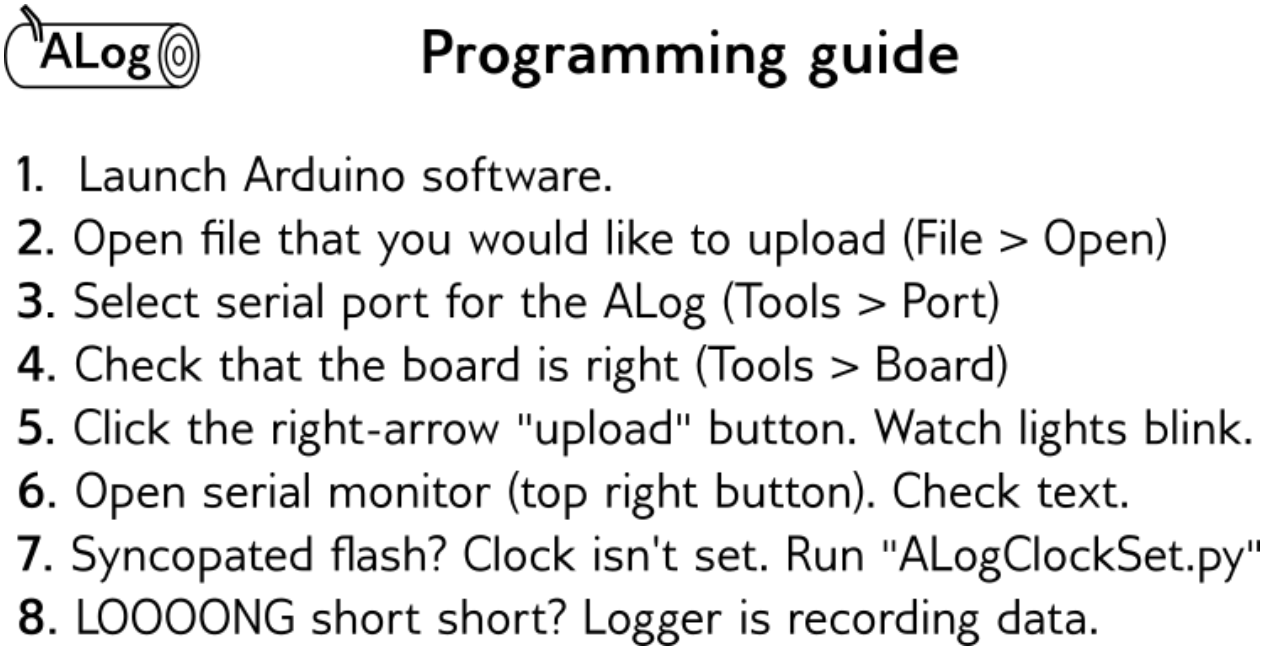


Figure 1.1 Quick reference

Quick-start guide

No-frills, no-pictures, as quick as possible. All the same material is covered in more detail in the ∗∗"Complete guide: from the basics onward,"∗∗ ***below; look there if you get stuck***

1. Download and install the Arduino IDE from https://www.arduino.cc/en/Main/Software
2. Install all required software libraries: start the IDE, go to "Sketch --> Include Library --> Manage Libraries...", and search for and install the following:
   (a) The ALog library; you can find this more easily by typing "Northern Widget" into the search box
   (b) The DS3231 library (for the real-time clock); this can also be found by typing "Northern Widget"
   (c) The SdFat library (for the SD card); simply typing "sdfat" into the search box will do.
   (d) The BMP180 library from SparkFun electronics (**key difference:** This must be downloaded and installed separately via a copy/paste into your Arduino/libraries folder; we are looking into options to avoid this necessity)
3. Add support for the ALog boards. (Skip this step if you're using a non-ALog Arduino.) Detailed instructions, available from https://github.com/NorthernWidget/Arduino_Boards, are also included below in the Complete Guide.
   (a) Go to File –> Preferences (Arduino –> Preferences on Mac) and paste this URL into the "Additional Boards Manager URLs" entry zone, in the lower right: https://raw.githubusercontent.com/↩ NorthernWidget/Arduino_Boards/master/package_NorthernWidget_index.json
   (b) Go to Tools –> Boards –> Boards Manager; type in "Northern Widget" and install these boards definitions.
4. Choose the board that you will be using. Go to tools –> board, and then (most likely) –> ALog BottleLogger v2.

5. Slide an SD card into the receptacle on the ALog BottleLogger.
6. Using a USB cable, plug your ALog data logger (or compatible Arduino device) into the computer
7. Go to File (Arduino on Mac) –> Examples –> ALog –> BasicStart. Click on it to load the file.
8. Upload the code to the logger: click on the "upload" button (right arrow) or press CTRL+u (command+u on Mac)
9. Look at the logger. You should see the blue and yellow lights flashing to show that it is communicating with the computer.
10. Open the serial monitor (top right button) and set the baud rate to 38400 bps.
11. See if the logger works; if it does, you are ready to start adding commands to read sensors.
    (a) If it doesn't, please check your progress through these steps, look through the rest of this guide, and if you really can't figure it out, email us at info@northernwidget.com.
    (b) Use the "RESET" button if at any point you need to restart the program
    (c) Use the "LOG" (or "LOG NOW") button to instantly take a reading, even if the logger would not otherwise be recording anything at that time

Basic Reference

For those already familiar with Arduino and with all of the required software already installed

Using the ALog library

Structure of an Arduino sketch with the ALog library

The following code is an Ardiuno "sketch" that includes the Alog library. It works whether you are using an:

- ALog data logger
- Arduino Uno (with ALog shield or equivalent clock and SD card)
- Arduino Mega (with ALog shield or equivalent clock and SD card)

Support for ARM-series Arduino boards is currently under development.

```
#include "ALog.h"

ALog alog;

// Note: Serial baud rate is set to 38400 bps

// USER-ENTERED VARIABLES //
char* dataLoggerName = "T01"; // Name of logger; displayed in Serial communications
char* fileName = "T01.txt"; // Name of file for logged data: 8.3 format (e.g,
                            // ABCDEFGH.TXT); <= 8 chars before ".txt" is OK

//Setup logging interval here, may set up more than one variable.
//Minimum interval = 1 sec, maximum interval is 23 hours, 59 minutes, 59 seconds
//0 for all means that the logger will not sleep
int Log_Interval_Seconds = 10; //Valid range is 0-59 seconds
int Log_Interval_Minutes = 0; //Valid range is 0-59 minutes
```

```
int Log_Interval_Hours = 0; //Valid range is 0-23 hours
// external_interrupt is true for a tipping bucket rain gauge
bool external_interrupt = false;

void setup(){
  alog.initialize(dataLoggerName, fileName,
    Log_Interval_Hours, Log_Interval_Minutes, Log_Interval_Seconds,
    external_interrupt);

  // If you are using a standard Arduino board (i.e. not a full ALog data logger)
  // and are not using the Arduino shield, you will have to set the proper pins for
  // the indicator LED (defaults to 9) and the SD card and RTC power (default to -1
  // to be inactive in the case of constant power supply; set these to the same
  // value if there is just one switch for both of these).
  // Replace "_pin" with your desired pin number, and uncomment the relevant line(s).
  // set_LEDpin(_pin);
  // set_SDpowerPin(_pin);
  // set_RTCpowerPin(_pin);

  alog.setupLogger();
}

void loop(){
  // ***************************************************

  alog.goToSleep_if_needed(); // Send logger to sleep
  alog.startLogging(); // Power up all systems, check WDT, reset alarms,
                       // and open data file(s) in write mode

  // ************ DO NOT EDIT ABOVE THIS LINE ************

  // READ SENSORS; GATHER/PROCESS DATA: EDIT THIS PART //

  alog.sensorPowerOn();

  // Turn on external power (3.3V and 5V in the case of the ALog BottleLogger)
  // for sensors and any other devices.
  // Place commands for all sensors that require this between
  // SensorPowerOn() and SensorPowerOff().
  // If you have no sensors that require power, you should comment out the
  // SensorPowerOn() and SensorPowerOff() commands.

  // !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!! //
  // INSERT COMMANDS TO READ SENSORS THAT REQUIRE ALOG-SUPPLIED POWER HERE! //
  // !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!! //

  alog.sensorPowerOff();

  // !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!! //
  // INSERT COMMANDS TO READ SENSORS THAT DO NOT REQUIRE ALOG-SUPPLIED POWER HERE! //
  // !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!! //


  // NOTE: THE BUFFER SIZE IS 512 BYTES;
  // run "alog.bufferWrite()" between your commands in this section
  // if you think you are approaching this limit.
  // Otherwise, the buffer will overflow and I'm not sure what will happen!
  // alog.bufferWrite()

  // ************ DO NOT EDIT BELOW THIS LINE ************

  // Wrap up files, turn off SD card, and go back to sleep
  alog.endLogging();

  // ***************************************************
}
```

Sensor commands

Sensor commands are defined in detail in the extended help, available below (PDF version) and at http://northernwidget.github.io/ALog/classALog.html.

In short, they look like:

```
alog.sensorName(parameter1, parameter2, parameter3)
```

where these parameters can be the pin numbers to which the sensors are attached, sensor-specific electrical characteristics (e.g., resistance), calibration values, and quantities that affect how the sensor behaves during its operation.

One example, for a thermistor (a temperature-sensitive resistor), is:

```
logger.thermistorB(10000, 3380, 10000, 25, 0);
```

Where the parameters are, in order:

- **R0**: Known resistance at given known temperature (here 10,000 ohms)
- **B**: The parameter governing the curvature of the resistance-to-temperature calibration curve (here 3380 K)
- **Rref**: The resistance of the known resistor (here also 10,000 , to optimize measurement sensitivity at 10 k)
- **T0degC**: The temperature [in Celsius] at which the known resistance equals **R0** (here 25℃)
- **thermPin**: The analog pin number to be read

This thermistor example is reiterated and expanded upon in the "Guide for first-time users: from the basics onward", below.

Adding support for new sensors

Printed below is the template function designed to guide users about how to add support for additional sensors. You may also look at ALog.cpp and ALog.h for our current examples, and feel free to contact us (info@northernwidget.↩ com) if you have questions about how to properly incorporate new sensors.

If you do add your sensor to our library or make an improvement, **we would really appreciate it if you would contact us about including the changes you've made**. We have designed the ALog library as a resource for the community, and the more of us who make it better, the bigger and better open-source field instrumentation grows!

```
void ALog::_sensor_function_template(uint8_t pin, float param1, float param2
      ,
           uint8_t ADC_bits, bool flag){
```

- alog.Example(A2, 1021.3, 15.2, True);
- “‘
- ∗/

float Vout_normalized_analog_example = analogReadOversample(pin, \ ADC_bits) / 1023.;

float Some_variable = Vout_normalized_analog_example ∗ param1 / param2; if (flag){ Some_variable /= 2.; }

//////////////// // SAVE DATA // ////////////////

if (first_log_after_booting_up){ headerfile.print("Some variable [units]"); headerfile.print(","); headerfile.sync(); }

// SD write datafile.print(Some_variable); datafile.print(F(","));

// Echo to serial Serial.print(Some_variable); Serial.print(F(","));

} “‘

LED

(This is identical to "LED messages", below in the full guide)

These messages are flashed on the large red LED. Those in bold are those that you may see most commonly.

- Syncopated: **daaaa-da-daaaa-da-daaaa-da**: Clock has reset to the year 2000! Please set the clock.
- **LONG-short-short**: All is good! Starting to log.
- 5 quick flashes: Missed first alarm; caught by backup alarm. Rebooting logger.
- **20 quick flashes**: SD card failed, or (more likely) is not present. This also happens if it is not seated properly; reseat the card and try again.
- 50 quick flashes: you have tried to reassign a pin critical to the ALog to another function; this will most likely break the system. Check your code and re-upload.
- 100 quick flashes: your Arduino model is not recognized by the ALog library!

Buttons

(This is identical to "Buttons: RESET and LOG", below in the full guide)

There are two buttons on the ALog:

- **RESET** starts the program over from the start. It's similar to "reboot" on your computer.
- **LOG** takes a reading at the exact instant it is pressed and causes the red indicator LED to strobe once.

Main Developers and Contact

The ALog has been developed by Andy Wickert and Chad Sandell at Northern Widget LLC and the University of Minnesota.

For questions related to the Logger library, please send a message to us at info@northernwidget.com.

Complete guide: from the basics onward

Are you new to the ALog, Arduino, and/or C/C++ programming? If so, this page is for you. We'll guide you through the steps to install a first simple program on your ALog data logger, and introduce you to a few concepts along the way.

Materials Needed

For the basics:

- **1x ALog data logger**
- **1x USB cable that fits the ALog's USB port**
  - USB A-B for version 2.1.0 and prior
  - USB A-miniB (like an Android cell phone) for version 2.2.0 and later
- **A computer with a USB port**

For the data logging exercises:

- **1x thermistor**
  - Our instructions are for the **CanTherm CWF1B103F3380** (available on Digi-Key or from us). This thermistor is 1 meter long and coated in epoxy, making it a good "out-of-the-box" tool, though we recommend calibration if you are going to try to make measurements with greater than factory precision (±1% of nominal resistance value, here 10 kilo-ohms (k) at 25 ℃).
- **1x reference resistor**
  - We typically use the **Vishay Dale PTF5610K000BYEB** for their high precision and temperature stability at a reasonable cost. Note that this factory precision is compounded with the precision of the thermistor when calculating error!
- **1x screwdriver**: Slotted. We recommend a 0.4 x 2 mm blade. Good options are this one and its ESD-safe version.

For the field

- ∗∗"The Basics"∗∗ (above):
  - **ALog data logger**
  - **Computer**
  - **USB cable**
- Any **sensor(s)** you need
- **Battery pack** with power cable attached; solar panel optional; see "Power" section for more details.
- **Housing** with any required holes drilled, cable glands and associated waterproofing, and (recommended) attachment points for logger and power supply; see "Housing and waterproofing" section for more details.
- **Tools**: my personal toolkit typically includes:
  - Screwdrivers (2-3): I recommend one standard larger one with both a Phillips head and a flat head (slotted) for general-purpose use and a small 0.4 x 2 mm blade slotted one for the loggers. Good options for the latter are this one and its ESD-safe version. If you have lots of hose clamps, a nut driver becomes a great asset.
  - Angle cutters
  - Wire stripper

- Spare wire
- Multimeter
- Adjustable wrench (mine in the photo is larger than is needed)
- Socket wrench (can be optional if you know that you won't need it)
- Hex wrenches (Allen wrenches / church keys) (I have English and Metric sets, but don't always bring both)
- Portable soldering iron: battery-powered (you get maybe 15-20 minutes on a 4x AA model) or butane
- Lead-free solder
- Electrical tape
- Hose clamps
- Permanent markers (I like multiple sizes of Sharpies)
- Cable ties

\

Figure 1.2 Andy's tool roll

A few definitions

- A **microcontroller** is a teeny tiny computer that has its own:
  - Processor for performing computations
  - Memory for storing programs and variables
  - Metal "pins" to connect to and interact with other devices (see immediately below)

  - An oscillator, such as the ALog BottleLogger's 8 MHz crystal, sets the speed of the microcontroller's computation. This means that the ALog BottleLogger is an 8 MHz computer.

- A **pin** is one of the "legs" on the microcontroller chip. Electrical power or signals flow through these. Many of these are connected to the **screw terminals** on the ALog board.

- `C` and `C++` are **programming languages**.

  - Programs written in these languages must be **compiled**. This means that they are turned into bytes (1s and 0s) by another piece of software before being made executable and (in our case) uploaded to a small board.

- **Arduino** is a software and hardware standard for microcontroller projects.

  - They are best known for the Arduino Uno board (and its predecessors).
  - Also important – and used by us – are their extensive software libraries to assist us in programming AVR microcontrollers.
  - **AVR** is the family of microcontrollers most commonly used by Arduino projects.

- The **ALog BottleLogger** is a lightweight and inexpensive low-power open-source data logger that incorporates elements of the Arduino system, and can be programmed through the Arduino IDE.

Looking at your board:

Your ALog board should look something like this. It has a lot of components. We'll start to look at them in more detail once you wire the board up.

\

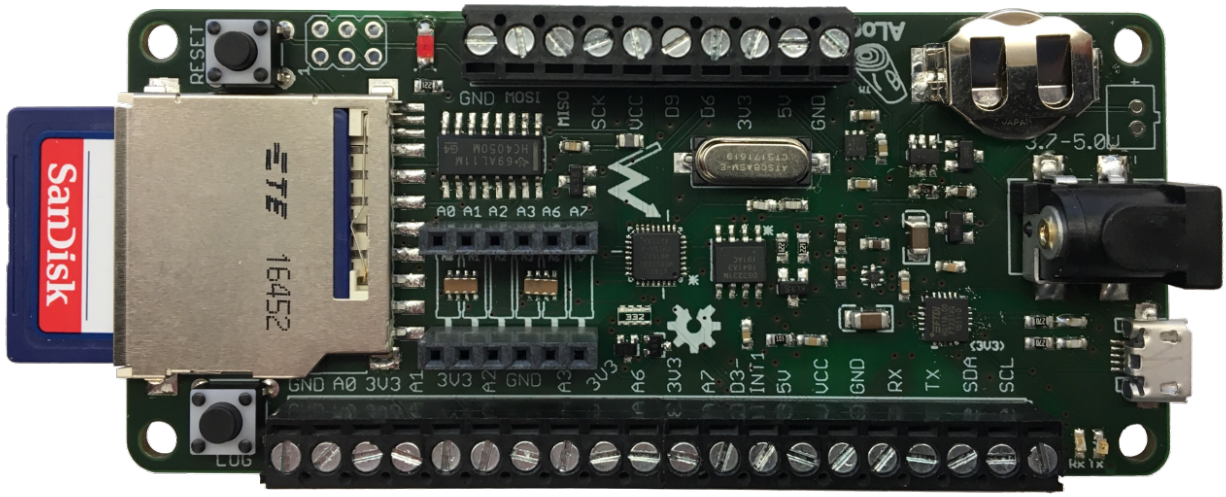


Figure 1.3 ALog BottleLogger

ALog BottleLogger v2.2.0 top photo

Dimensions: namesake

The dimensions of the ALog BottleLogger are perfect to slip it into a Nalgene water bottle: these are often easier to find than gasketed boxes, especially if you're in the outdoors (or outdoors shops).

"Chunky" design for the field

Yes, we use the big, chunky, old-school SD cards, USB ports, and barrel jack connectors. And we love 'em. Why? Have you ever tried to connect cables with mittens in Minnesota-frigid temperatures? Have you ever dropped a micro-SD card (or other object the size of your pinky nail) into a beautiful bed of colorful oak leaves, stand of prairie grasses, or a lake? You'd better hope that your missing item is as big and recognizable as possible. We call it designing for reality.

(An update... while we love the old-school cool of the big USB A-B cable, we\ couldn't help but notice that tons of us were traveling with A-micro-B cables\ thanks to Android phones, but had to make a special effort to remember to bring\ our A-B cables to the field. So we've made a recent (2017) update to the USB\ port in our design.)

What does it have?

- A teeny tiny computer running at 8 MHz
- An SD card to save data
- A real-time clock and its backup battery to save the time
- A USB port to upload programs and talk to the computer
- A bunch of "screw terminals" to connect wires for sensors, etc.
- Six pairs of holes for "reference resistors" to act as standards for certain types of measurements ("analog")
- A blinky light to tell you what's going on

Pins

The metal connections between the microcontroller (teeny tiny computer) and any external component of the system (e.g., power supply, sensors) are called "pins".

You are not attaching anything directly to the pins on the integrated circuit, obviously; your interface to the ALog data logger will be primarily through the screw terminals (which in turn are discussed immediately below) at the edges of the board. I refer to each screw terminal as a "pin" or "port", or simply as a "screw terminal"; there is no hard-and-fast language for this, except insofar as only those microcontroller pins that are exposed to the outside world via screw terminals are called "ports" in this documentation.

The Alog BottleLogger follows the Arduino Uno convention for pin numbering.

More information on pins can be found towards the end of this document, in the "Pin definitions" section.

Those little numbers and letters by the screw terminals

Those black things on the sides with screw heads are the "screw terminals". They clamp down on wires to connect peripherals to the ALog BottleLogger, securely. There are several different sets of letters by them. These are:

VCC

"Voltage of the common connector": this is raw power from the batteries. Keep it above 3.6V and below 5.5V. Any higher and you fry things; any lower and the system starts shutting down. We usually use:

- USB (5V)
- 3x AA or D batteries

5V

This is 5V power, supplied by a switchable charge pump. Note that it will be closer to 5.2V if very small voltages are required. It is most often used to power 5V devices.

3V3

This is 3.3V power, supplied by a switchable voltage regulator. It powers 3.3V devices and is used as a reference (and often a power source) for analog measurements.

GND

This is ground. Circuits that start at VCC end here. Don't touch power directly to ground unless you want to destroy something. It is the same as 0V.

A0, A1, A2, A3, A6, A7 (Analog pins, ALog BottleLogger)

These are the analog pins. They measure input voltages in a range between 0 and 3.3 volts. What happened to 4 and 5? They are talking to the real-time clock, and are labeled SDA and SCL on the board.

All analog pins can be used as digital pins as well!

When writing code, the analog pins can be referred to as A0, A1, A2, A3, A6, or A7 safely in all cases.

∗Additional <info:∗>

The analog pins can be referred to by their base numeral (for the ALog BottleLogger, 0, 1, 2, 3, 6, or 7) in the `Analog↩ Read(analogPinNumber)` *command that we'll see below; this is the legacy of early versions of Arduino. They must be elsewhere referred to as A0, A1, A2, A3, A6, or A7.*

A bit of advanced knowledge: "A0" maps to "14", "A1" maps to 15, and so forth until A7 maps to 21; this is because there are 14 digital pins (0-14), so the analog numbers come right after these. Therefore, while I find A0...A7 easier to remember, you may prefer to use the numbers >13, which is equally valid.

D3-INT1, D7, D8 (digital pins, ALog BottleLogger)

These pins can read or write TRUE or FALSE values. For writing, TRUE means full power (VCC) is delivered. False means the pin goes to GND. Reading values with these pins tells you whether the observed voltage is closer to VCC or to GND.

When programming these pins, refer to them by their number alone, dropping the D (and anything after the number that immediately follows the D.

D3-INT1 (interrupt, ALog BottleLogger)

This pin can be used as an interrupt, to observe an event. We often use it with anemometers (spin counters) and tipping-bucket rain gauges.

When programming with this pin as an interrupt, use the following syntax:\ `attachInterrupt(digitalPinTo↩ Interrupt(3), ...)`

If you insert the interrupt number (1), this will correspond to Pin 3 on the Arduino Uno or Alog BottleLogger, but will refer to Pin 1 on ARM-based devices, so this will cause compatibility problems.

D7 and D8: (PWM, ALog BottleLogger)

Pulse-width modulation. Actually, this isn't written on the board. It uses the analogWrite() function in Arduino and simulates a voltage between GND (0V) and VCC by turning itself on and off very quickly, repeatedly.

Rx and Tx

Receive and transmit: these are your serial pins. Be careful: they are also attached to the USB port (to talk to your computer), so they must be shared. For the electronics-savvy: they are connected to the computer via 1 k in-series resistors, so external sensors will override USB communications. See "UART", below.

MOSI, MISO, SCK

These are the pins for the SPI communications protocol. This protocol is used to communicate with the SD card. You may also use it with some external sensors. See section "SPI", below.

SDA, SCL

These are the pins used for the I2C communications protocol. This works with the real-time clock, and can connect to external peripherals as well. For the tech-heads: we are updating the design to optimize this for 3.3V design, common among I2C devices. See section "I2C", below.

Special pins

The ALog BottleLogger has several pins that are used for dedicated functions. These are:

- **Pin 2:** Interrupt connected to clock: triggers logger to wake
- **Pin 4:** External (sensor) power 3.3V and 5V regulator on/off switch
- **Pin 5:** The "LOG" (also referred to as "LOG NOW") button uses this pin to tell the logger to wake up and take a reading immediately.
- **Pin 7:** Clock and SD card power on/off switch
- **Pin 8:** Message-flashing LED on/off
- **Pin 10:** The chip-select pin for the SPI communication to the SD card; more on this in the "SPI" section, below

In addition, there are three significant digital interfaces that have uses internal to the ALog but that may also be used to communicate with external sensors:

UART

The UART, or universal asynchronous receiver–transmitter, communicates to the computer via a USB–serial converter (model number FT231X). It transmits these signals through in-series 1 k resistors that act to limit the current that passes in this signal, with the goal of allowing external sensors that communicate via the UART to take precedence over USB–serial communications to the computer. While this is available and functions, we recommend against its use as there will almost certainly be some amount of interference between the communications with the computer and with the sensor. In-development versions of ALog boards will have multiple UART ports, and these will include a dedicated port for communication with the computer and other UART ports for use with sensors.

- **Pin 0:** Rx (external device sends, logger receives)
- **Pin 1:** Tx (logger sends, external device receives)

SPI

SPI is a 4-wire communication protocol (i.e., bus): three wires/pins are shared between all SPI devices, while a fourth one is used as a "chip select" pin that is set LOW for communication with the logger and HIGH when all communications should be ignored.

It is straightforward to extend SPI to other devices by configuring one of the pins on the ALog data logger to be the new "chip select" pin. In addition to this, the pins with a set configuration are:

- **Pin 11:** MOSI (logger sends, external device receives)
- **Pin 12:** MISO (external device sends, logger receives)
- **Pin 13:** SCK (serial clock)

The abbreviations stand for "Master out, slave in", "master in, slave out", and "serial clock". I'm really not OK using the term "slave" for about a million reasons, so whenever I need to refer to the device attached to the logger, I will write "agent" or "external device". This goes for any bus – not just SPI.

Additional information on SPI and setting up additional SPI devices may be found at:https://www.arduino.cc/en/reference/SPI.

I2C

I2C is a 2-wire communications protocol. It is used to communicate with the real-time-clock (RTC, model number DS3231), as well as many sensors. Each attached device has its own address, enabling **up to 128 sensors** to be attached to the same I2C port as the clock. But just be careful that your sensor code is good – if it is not and halts the I2C system, it could affect the clock and therefore the entirety of the operations of the ALog!

- **Pin A4** (= pin 18): SDA (Serial Data)
- **Pin A5** (= pin 19): SCL (Serial Clock)

Reference resistor headers

The pair of plastic vertical "headers" – things with holes in them for wires – are for reference resistors to read analog resistance-based sensors. A simple example is a thermistor, and we'll cover that farther on down here. For now, just consider these resistors to be standards against which many analog sensors will be compared to measure their resistance values.

Buttons: RESET and LOG

There are two buttons on the ALog:

- **RESET** starts the program over from the start. It's similar to "reboot" on your computer.
- **LOG** takes a reading at the exact instant it is pressed and causes the red indicator LED to strobe once.

Programming Arduino-based systems: Focus on the ALog

Download and install the Arduino IDE

The Arduino IDE (Integrated Development Environment) is the current programming environment we use to write and upload programs to the ALog. (Other options exist, but this is the most beginner-friendly.) We haven't yet tested the brand-new web editor, so we'll be suggesting an old-fashioned download. And if you're deploying these in the field, you'll need the downloaded version! Go to https://www.arduino.cc/en/Main/Software. Get it, install it, go.

Set up the ALog boards definitions

The ALog boards are "third-party" Arduino boards, so you'll have to set up support for these boards yourself. We've made a pretty thorough walkthrough that you can view here at https://github.com/NorthernWidget/↩Arduino_Boards, and have included those installation instructions here, immediately below.

Each board will be added as an entry to the Arduino **Tools** > **Board** menu.

Go to **File** > **Preferences** (or **Arduino** > **Preferences** on Mac).

\

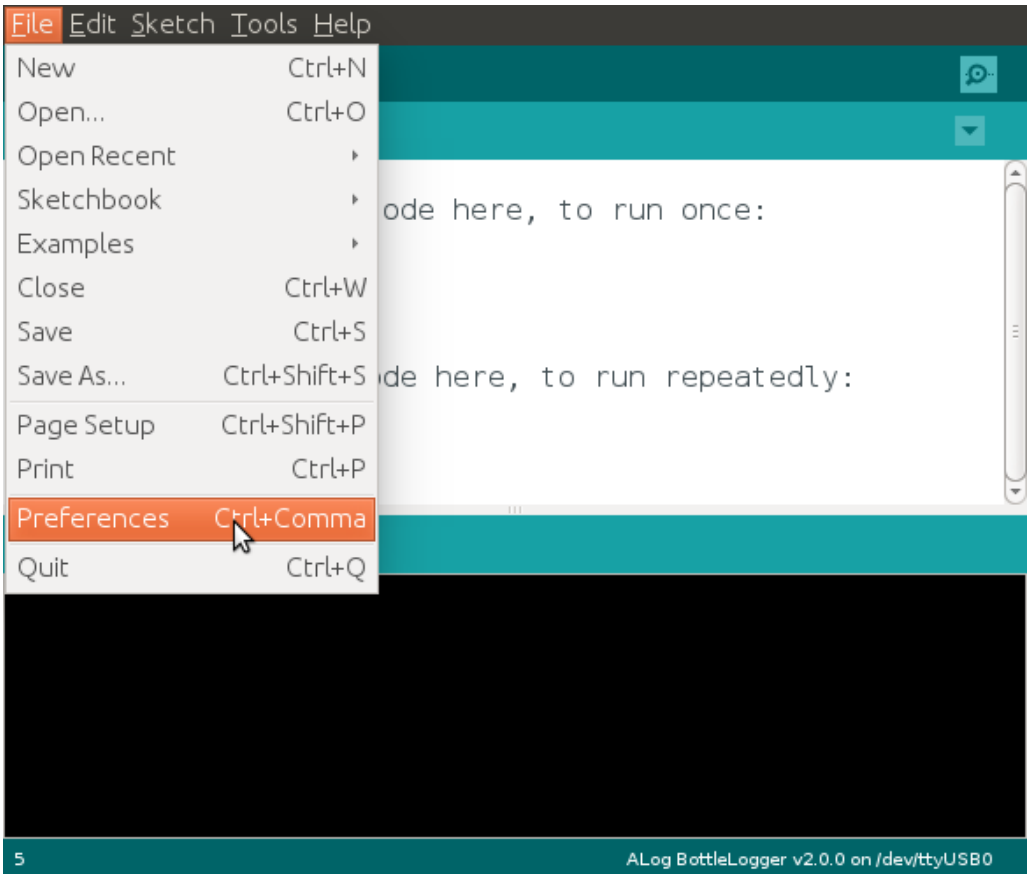


Figure 1.4 Arduino IDE

preferences"

Open the Arduino IDE preferences

Open 'Additional Boards Manager URLs', and paste the following in either the box for **Additional Boards Manager URLs**, or, if this is populated, the window that pops up when you hit the button to the right of the **Additional Boards Manager URLs** text entry area:

```
https://raw.githubusercontent.com/NorthernWidget/Arduino_Boards/master/package_NorthernWidget_index.json
```

\

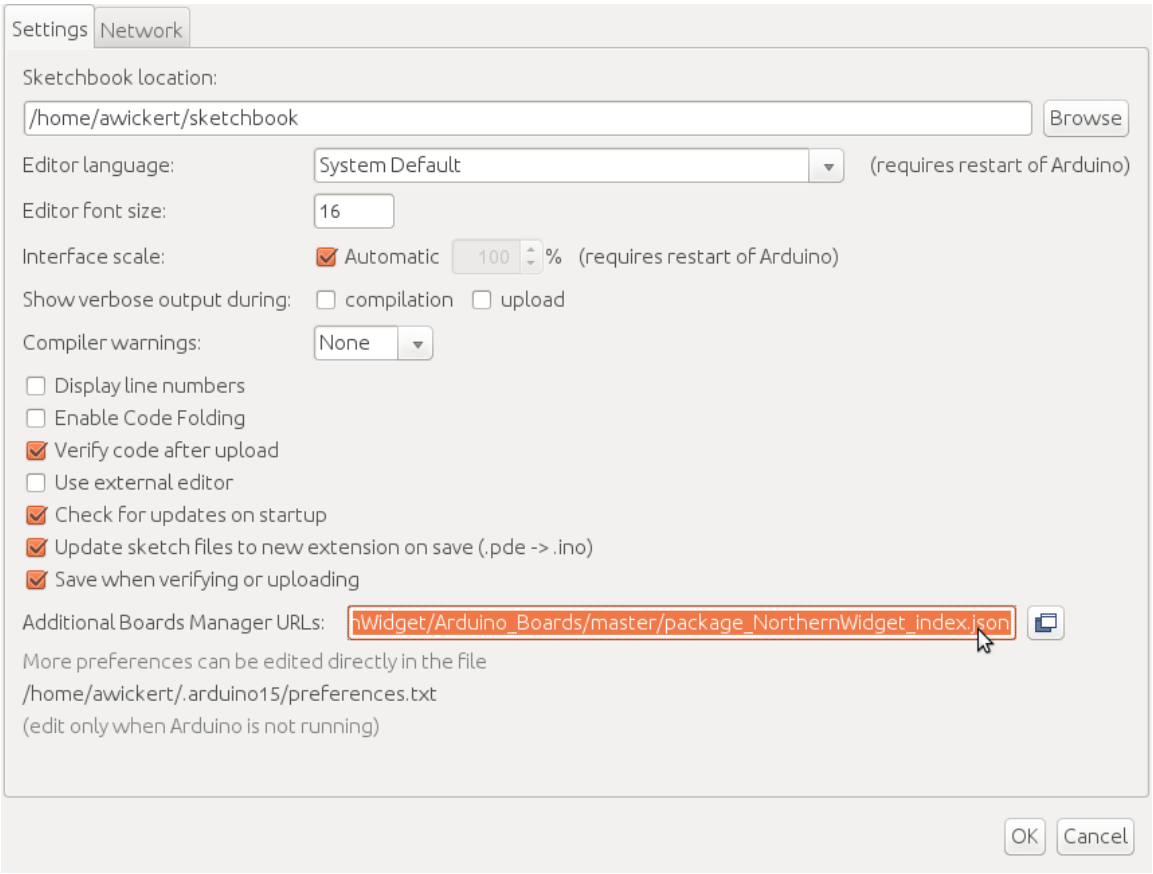


Figure 1.5 URL here

Paste the URL here.

![Unless you already have done that for third-party boards... in that case, open this frame and paste the URL here.](https://github.com/NorthernWidget/ALog/raw/master/doc/figures/Arduino_Boards/BoardURLs_list.png "Unless you already have done that for third-party boards... in that case, open this frame and paste the URL here.")\

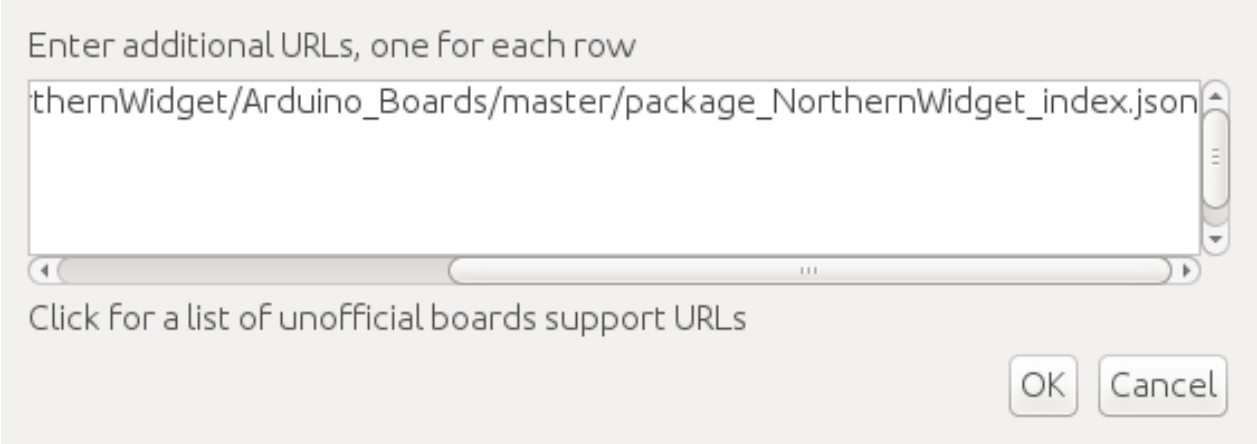


Figure 1.6 Or here

Unless you already have done that for third-party boards... in that case, open this frame and paste the URL here.

Now, go to **Tools** > **Board** > **Boards Manager...**.

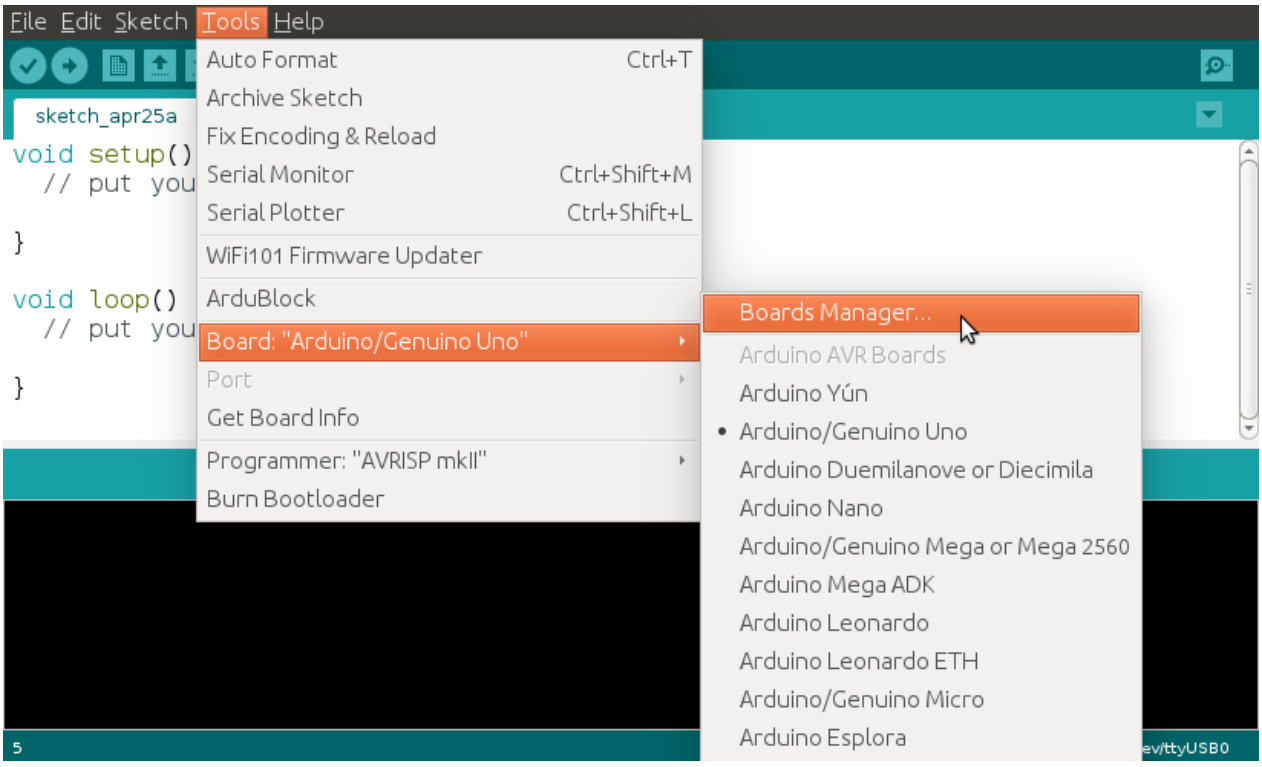


Figure 1.7 Boards

Open the boards manager here.

Click it, and the following window will appear:

\

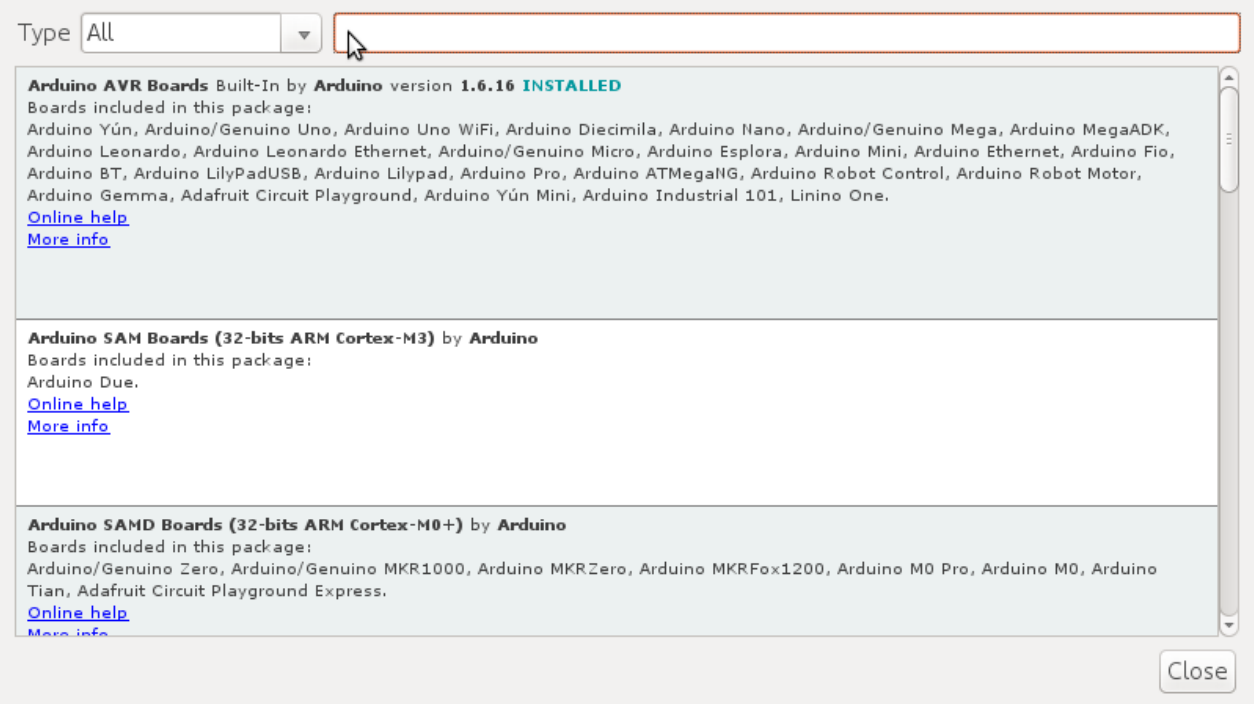


Figure 1.8 Boards manager

Boards Manager.

if you type in "Northern Widget" (or usually just "Northern" as well), you should see an option to install board files for Northern Widget Arduino compatible boards.

\

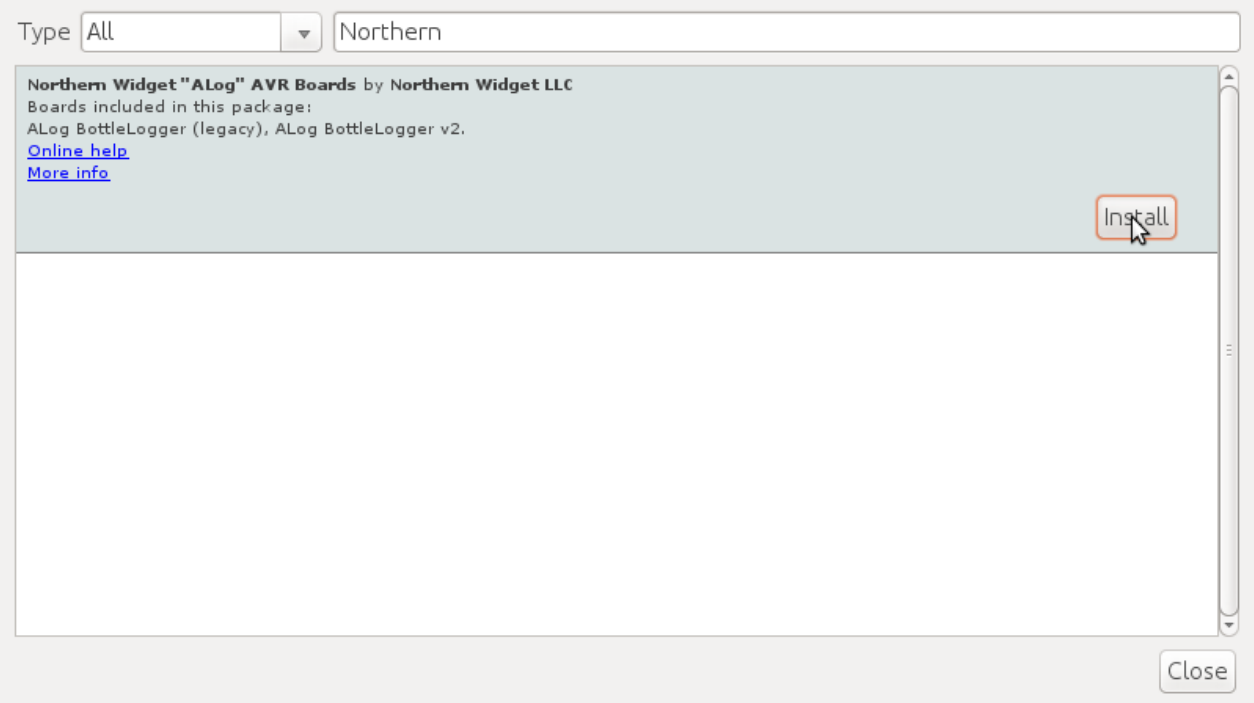


Figure 1.9 Boards

Northern Widget boards.

If Northern Widget options do not appear. restart your Arduino IDE and try again.

Click "Install" to add the NorthernWidget boards to your list. At the time of writing, we support only AVR boards, but this may change soon.

![ALog (AVR) support installed.](https://github.com/NorthernWidget/ALog/raw/master/doc/figures/Arduino_Boards/BoardsManager_Northern_Done.png "ALog (AVR) support installed.")\

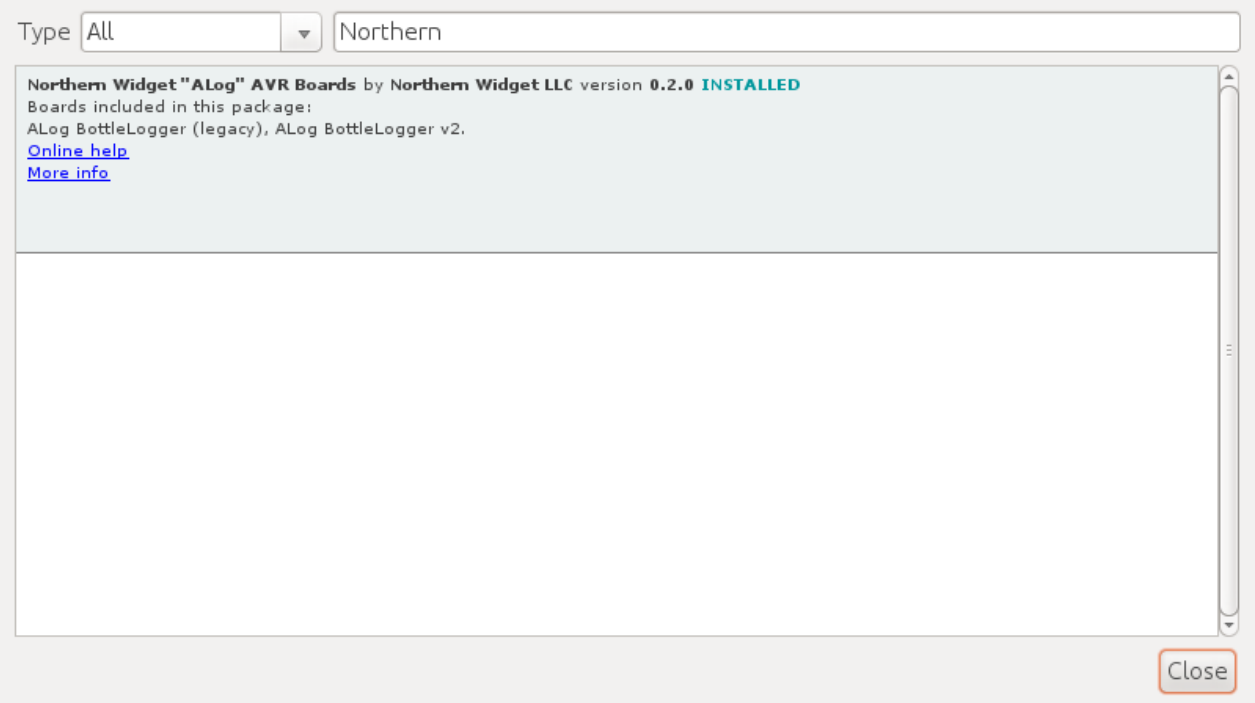


Figure 1.10 Done.

ALog (AVR) support installed.

Now, when you select the Boards list, you will see a collection of new boards for Northern Widget.

You will then want to change your chosen board to the board that you have. At the time of writing, this is probably the "ALog BottleLogger v2"; the "legacy" option is used for v2.0.0-beta and prior, and there are fewer of these boards in circulation.

\

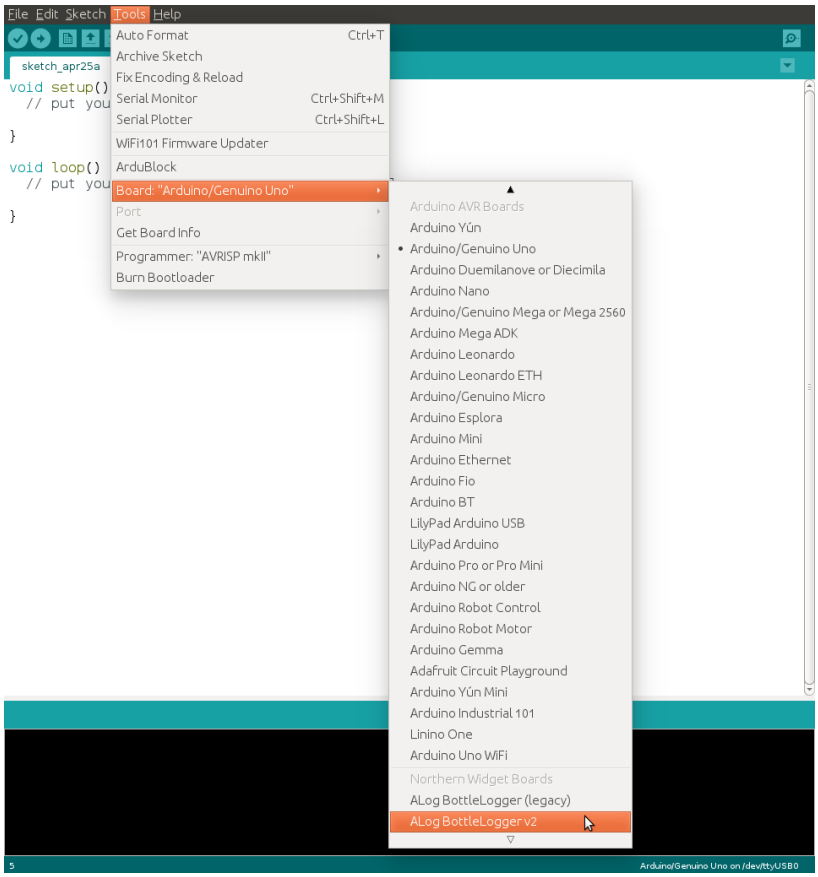


Figure 1.11 Select board

Select the proper board.

That's it! You're done, and ready to rock and roll... er, collect data, with your ALog data logger.

Download and install the custom ALog libraries

The ALog also relies on some custom software libraries. These are collections of computer code that help make everything from running basic data logging utilities to interfacing with specific sensors easier. We're introducing you to these libraries up here, even though they won't be needed until quite a bit further down. While we highly recommend them and they are very helpful for field work, you can also program the ALog without them! In fact, the earlier examples (below) do not include the custom libraries.

Option 1: downloading and installing through the Arduino IDE

Currently not recommended, at least until all components can be downloaded this way

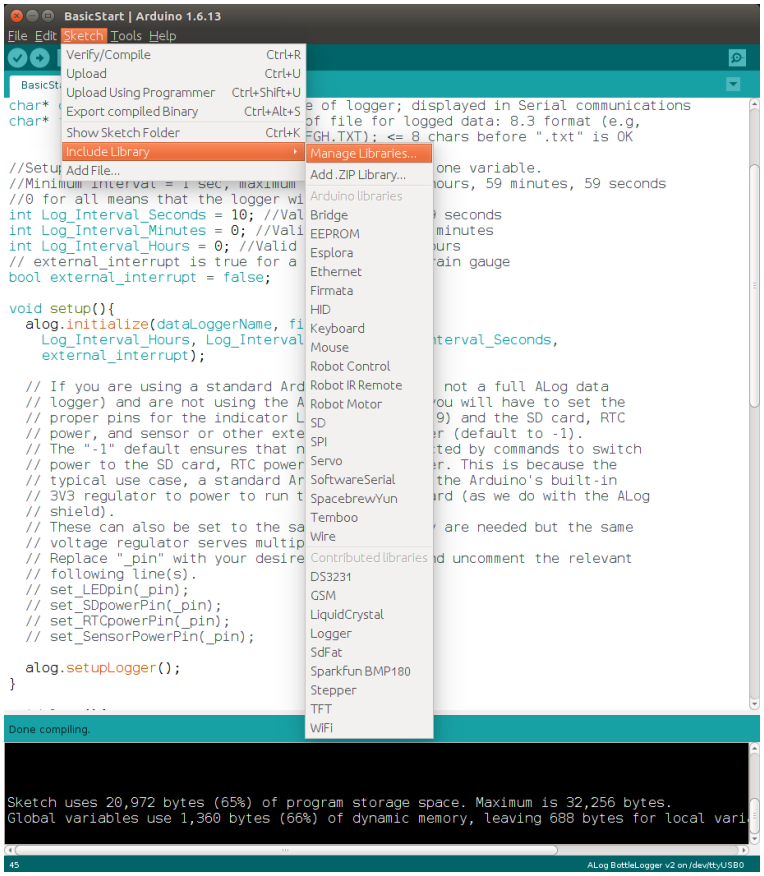
Stable versions, but outdated during times of rapid development

The Arduino IDE has a library manager! Here we will walk you through how to use it to obtain and install the requisite libraries. This uses the graphical interface, but can take longer than, "Option 2", below.

At the moment, you will also have to download the stable release (below) and copy the SFE_BMP180 library into your "Arduino/libraries" folder; we're working to streamline this.

First, open the libraries manager.

\



"libraries"

Then, look up each of the three core libraries:

DS3231 (real-time clock):

\

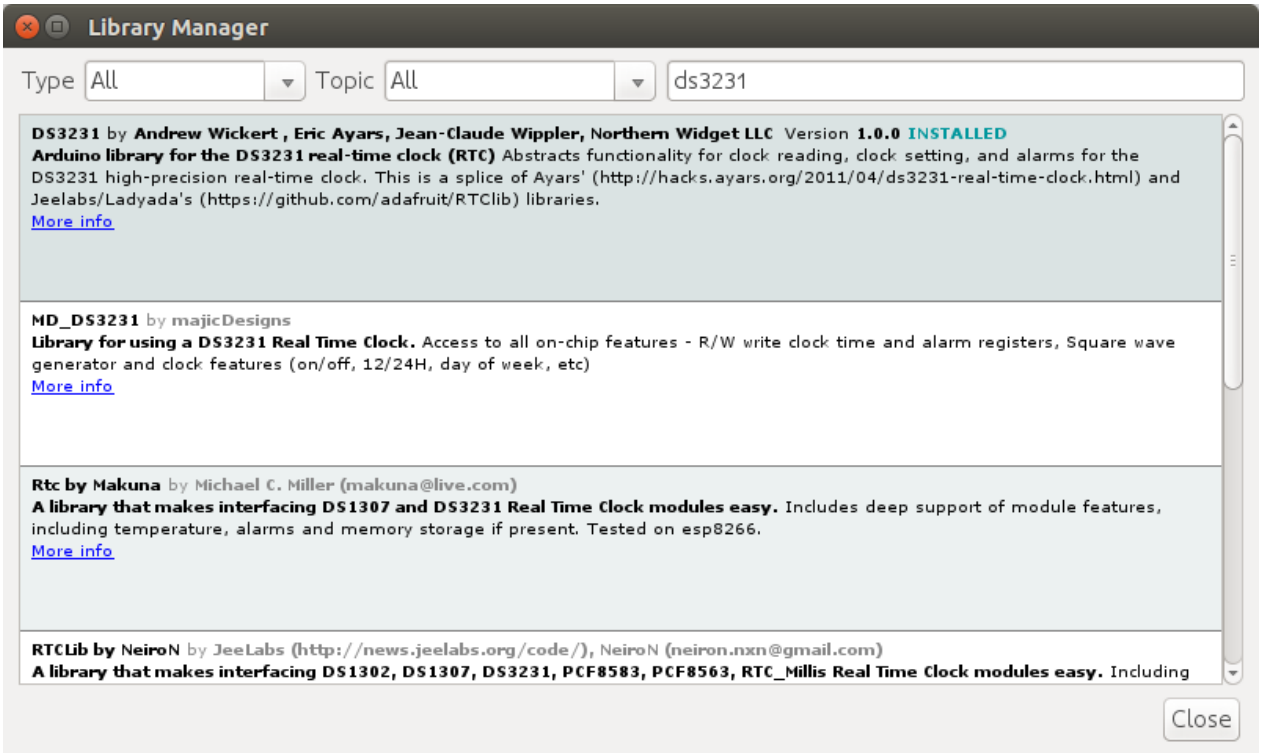


Figure 1.12 DS3231 library

SdFat (SD card):

\

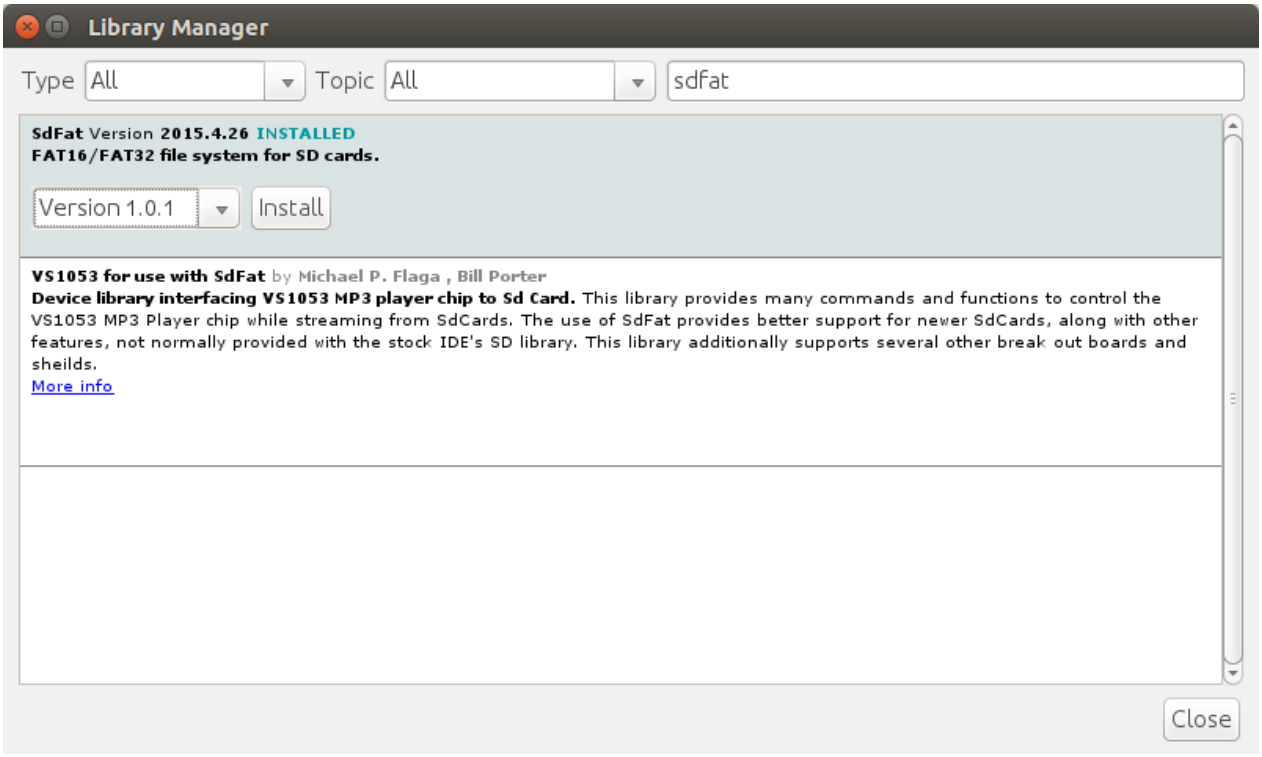


Figure 1.13 SdFat library

ALog (all data logging!):

\

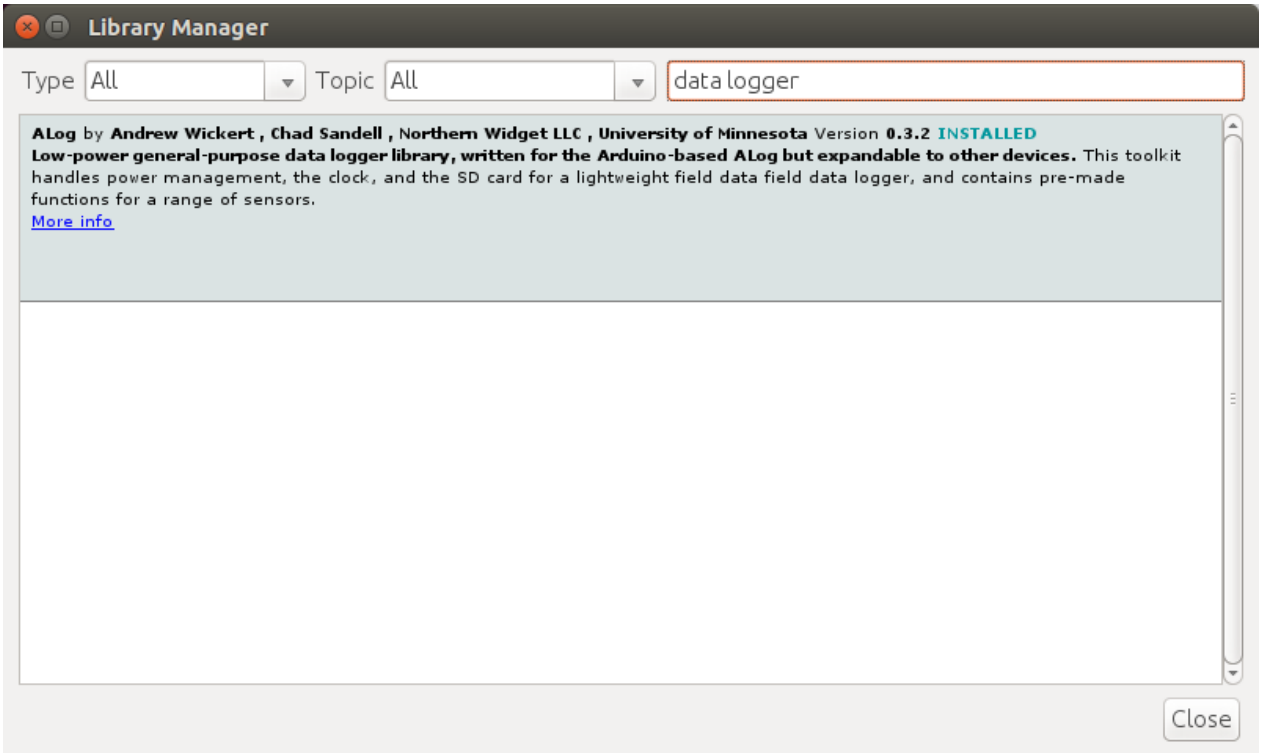


Figure 1.14 ALog library

Once this is done, go to the instructions for Option 2 (immediately below) to get the SFE_BMP180 library.

Option 2: direct installation from a prepared file archive

Not automated in the IDE but much simpler!

Download:

Stable release: `[ZIP]` `[TAR.GZ]` **(recommended)**

Nightly build: `[ZIP]` `[TAR.GZ]` (for only those who can tolerate things breaking once in a while)

Extract/Install:

Unzip the archive, and place the contents in your "∗∗Arduino/libraries∗∗" folder. This should be in your home directory or your Documents. Then restart the Arduino IDE (if it is open).

Learn some basics of programming in the Arduino language

If you're not a programmer (yet!), you might be thinking, "yeah, right, I can't program."

But we say, ∗∗"Yeah, right! You can program!"∗∗ We'll walk you through it:

- First, we'll point you to some nice resources
- Then, we'll show you how to write a simple code to blink your LED.
- Finally, you'll combine your LED blinking with serial communications with your computer.

∗[For the folks already in the know: the Arduino programming language is variant of C/C++.]∗

Programming resources

Tutorials from Adafruit

These are serious good. You can follow along with them! https://learn.adafruit.com/series/learn-arduino If you follow them, note that the ALog BottleLogger can take only 3.6 – 5.5 volts, and that the LED is controlled by Pin 8 (not 13). Here is their version of the "blink" example that we'll do below. https://learn.adafruit.com/adafruit-arduino-lesson-1-blink?view=all Look at as many of these as you want! Yes, your ALog can:

- Sense sensors
- Motor motors
- Blink blinky things
- Text text via the serial monitor
- Type text onto a display screen
- More more more!

The full reference guide

Yes, there **is** a manual! The full reference guide to programming Arduino devices is here https://www.arduino.cc/en/Reference/HomePage.

The internet

We're not kidding. We type things into our favorite search engine when we can't figure them out and/or are too tired to think. 90+% of the time, this gets us to the right answer. Sometimes it takes longer than others. Arduino forums and StackExchange are pretty great: **Arduino forums** Arduino stack exchange [stack exchange is just about the most effective way of conveying the right answers for how to do things that I've (Andy has, that is!) ever seen on the internet]

Basic syntax of C programming

Programming languages all have ways in which logical thoughts are placed into text that is entered into a computer. Here is a non-exhaustive list of these. It is not intended to be complete, but to give you some essential tools to start reading the code that appears below.

Defining variables

Each variable needs to be defined before it is used.

```
bool isOn=False; // Boolean: True (1) or False (0)
int x=50; // Integer (this works for values between -32768 and 32767).
          // For broader applicability, look up uint32, int64, etc.
float _my_float = 1000.293; // "Floating point" (decimal) number
char letter='h' // This is a letter, but is represented in ASCII, so is
                // also a 1-byte number
```

Each of these goes: **variable_type variable_name = variable_value ;** For more information on types, see the Sparkfun Electronics tutorial at https://learn.sparkfun.com/tutorials/data-types-in-arduino. For what you are doing right now, a general idea of what these definitions are and what they look like should work.

Comments

Ignore lines that start with:

```
// [compiler doesn't care what is here, but you can read it]
```

And lines that are between

```
/*
 whatevs
 */
```

These are for the benefit of the humans reading the code. Comment your code clearly and thoroughly!

Line endings

Lines must be terminated with semicolons.

```
int a=5;
```

Operations

Basic arithmetic operators ($+$, $-$, $*$, $/$) work. Division of two integers results in "floor division, or rounding down the result. There are also logical operators ($<$, $>$, $==$, etc.)

Loops and Statements

These are ways to control the flow of a code. I'm afraid that this introduction to the ALog would be diluted by the details of these. I point the interested reader to the main Arduino reference: https://www.arduino.cc/en/Reference/HomePage.

Connecting the ALog and Computer

Use a USB cable to connect the ALog to your computer. (Photo from ALog v2.0.0; v2.2.0+ use microUSB, like Android cell phones.)

![Insert USB cable (photo R. Schulz).](https://github.com/NorthernWidget/ALog/raw/master/doc/figures/InsertUSB.png "Insert USB cable (photo R. Schulz).")\

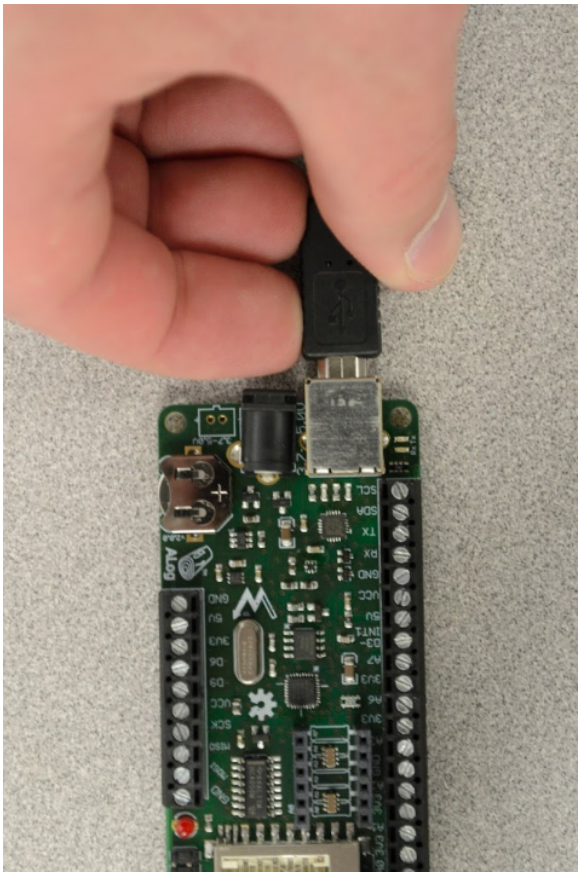


Figure 1.15 USB connection

Then, using the Arduino IDE, select the name of the USB-serial port that is connected to the ALog.

\

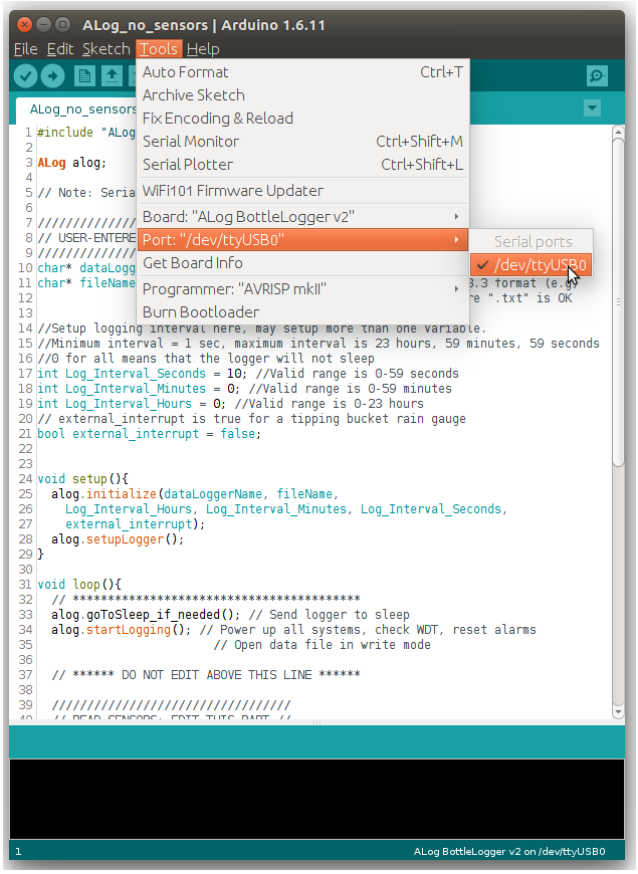


Figure 1.16 Serial Port

Now your ALog is ready for communications and programming.

Example Programs: Writing and Uploading

Arduino programs, often called "sketches", are how you tell the ALog data logger what to do. Here is some information to get you started.

A new program

If you open a new Arduino program, it will look like this:

```
void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:

}
```

`setup()` and `loop()` are both **functions**. These are specific sets of code that are executed when their names are called in the code. These two functions are special in that:

1. Every Arduino program much have them.
2. They do not need to be called by code that you write: Arduino automatically interprets and, indeed, requires that these be included.
    - Everything inside the curly braces after `setup()` is run once, when the data logger starts running
    - Everything inside the curly braces after `loop()` is run continuously, after `setup()`.

Before these functions, you can place all of your variable definitions.

Uploading code to the ALog data logger

Once your code is written – either as a copy/paste of this or as your own – save your code. All Arduino sketches need to be within their own folder. After this, you can hit the "upload" button (right arrow) to load the code to the board. (The check mark to the left will test if your code compiles.)

![Upload sketch (program) to board.](https://github.com/NorthernWidget/ALog/raw/master/doc/figures/ArduinoScreenshots/Uploading.png "Uploading sketch (program) to board.")\

Figure 1.17 Uploading to board

Blink

Let's make a blinky light turn on and off! On the ALog, **pin 8** is attached to a red indicator LED. Here is some code to create a syncopated blinky light.

```
// Syncopated LED blink

void setup() {
 // put your setup code here, to run once:
 pinMode(8, OUTPUT); // "Output" means that the maximum current possible will be available at Pin 8
                     // to drive the LED
}

void loop() {
 // put your main code here, to run repeatedly:
 digitalWrite(8, HIGH); // Turn power to light ON
 delay(750); // Wait for 750 milliseconds
 digitalWrite(8, LOW); // Turn power to light OFF
 delay(250);
 digitalWrite(8, HIGH);
 delay(250);
 digitalWrite(8, LOW);
 delay(250);
}
```

Look at the comments in the code for information on what each of the commands do; these are detailed more below in the USB/Serial communications section.

USB/Serial communications

A second important piece of your work with the Arduino is to communicate with it. We'll start with one-way communications, with the ALog talking to your computer. We will still flash the LED, and will add in information on how to define a variable outside of these two functions. The comments give you some general idea of what each section does; I will break down the importance of each section below.

Before running this code, open the serial monitor and ***change your baud rate to 38400 bps!***

\

Figure 1.18 Serial monitor

(Note that if I would have clicked, an error message would have popped up below saying that it cannot find a device at the given port: this is because I have no board plugged in. If you get this message and **do** have a board plugged in, go to **Tools** > **Port** > **[new port that you pick]**)

\

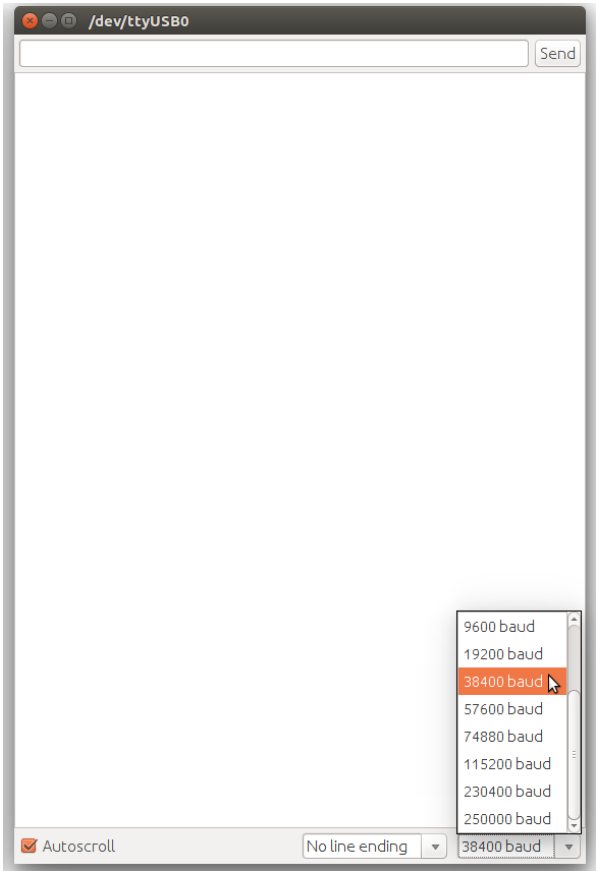


Figure 1.19 Baud

```
// Global variables
// These variables are available to any function inside your code
int LEDpin = 8;

void setup() {
  // put your setup code here, to run once:
  Serial.begin(38400);
  pinMode(LEDpin, OUTPUT);
}

void loop() {
  // put your main code here, to run repeatedly:
  Serial.println("Hi, computer.");
  delay(1000);
  digitalWrite(LEDpin, HIGH);
  delay(500);
  digitalWrite(LEDpin, LOW);
}
```

Let's break this down a little bit:

Global variables and pin definitions

```
// Global variables
// These variables are available to any function inside your code
int LEDpin = 8;
```

Here we are defining the pin number that controls the LED. This is Pin 8, which is a digital pin; analog pins would generally have an "A" before their pin number, and can also be used for digital input and output.

Setup

```
void setup() {
  // put your setup code here, to run once:
  Serial.begin(38400);
  pinMode(LEDpin, OUTPUT);
}
```

"**Serial**" is a library for USB-Serial communications between the computer and the attached Arduino. It is a core part of Arduino, and therefore does not have to be "included" at the top in the way the ALog library (farther below) must be. To view serial communications, you may open the serial monitor in the Arduino IDE (the figures immediately above describe how to do this).

Within the `Serial.begin()` command, we set the **baud rate** to "38400"; this is in bits per second. *Technical note: We choose this rate because it is closer to being integer-divisible by our 8 MHz processor clock speed than other baud rates are.*

the **pinMode** command can be set to **input** or **output**; "input" means that the pin is ready to read an incoming signal; output means that it maximizes the amount of electrical current it can source to send a strong output signal that is also enough (40 mA absolute maximum) to power basic devices.

Loop

```
void loop() {
  // put your main code here, to run repeatedly:
  Serial.println("Hi, computer.");
  delay(1000);
  digitalWrite(LEDpin, HIGH);
  delay(500);
  digitalWrite(LEDpin, LOW);
}
```

Serial.println prints the requested text and then moves ahead to a new line for the next print command (if any).

delay takes an argument in milliseconds that describes how long you would like the logger to do nothing but count time passing.

digitalWrite turns an output pin ON (HIGH) or OFF (LOW). When HIGH, the pin is sending current at a voltage corresponding to VCC (i.e., the positive voltage for the system). When low, it is sending current at a voltage corresponding to GND (ground / Earth).

Using the ALog library

The full reference to the ALog library is located at\ `http://northernwidget.github.io/ALog/classALog.html`.

The following sections walk you through the functionality.

ALog library components

The ALog library contains:

- **Global variables** that are important across the entire library, such as pins based on the design of the board that you are using and variables that determine how frequently the logger will read sensors;
- An **object-oriented interface** that allows the user to interact with the ALog library;
- **Utility code** that carries our behind-the-scenes work to keep the ALog running well; and
- **Sensor functions** that are pre-written and allow users to connect with sensors as part of a single-line function call.

Object-oriented interface

The end user can ***instantiate*** the ALog class, that is, create one's own version of it for a particular purpose, as follows:

```
ALog alog;
```

Here, "ALog" is the class, and "alog" is one's instance of the class. It's a bit like picking out a Lego set from a store: it becomes yours, and you can build what you like with it, within the constraints of the pieces that you have.

You may call any function within the ALog class by typing `alog.<function_name>(<parameters_to_pass>).`

Utility code (both "private" and "public" functions)

The utility code is the unseen backbone of the ALog library, and much of the reason that I wrote it in the first place. It handles power consumption, device stability, communications with the user, real-time clock management, and reading and writing files to and from the SD card.

Sleeping and waking

The sleep mode is the most important piece of the ALog data logger. It's the superlative of when you close the lid of a laptop computer: the ALog goes from drawing at least 6.7 milliamps to just 53 microamps – that's 0.053 milliamps, for a **greater than 100x increase in power efficiency**, and therefore battery life.

The internal sleep functions are activated by the end-user as part of this command:

```
alog.goToSleep_if_needed(); // Send logger to sleep
```

When the logger **wakes up**, it takes readings and then goes back to sleep. How does it wake up? There are two ways:

1. The real-time clock's alarm function triggers an interrupt.
2. An event activates a second interrupt and causes it to wake; an example is a bucket tip from a rain gauge.

The logger can also be run in a mode to **maximize logging speed** (around 10 Hz at top speed, but not well-measured at the time of writing) and never sleep!

Real-time clock

The real-time clock maintains the absolute time to within ±0.432 s/day. It also operates two alarms, which can be used to trigger the ALog.

As a safety feature, we enable the second of these alarms to go off several after the first one is triggered at the user-specified time. If the first alarm goes off, the second is disabled and advanced farther into the future. Otherwise, if the first alarm doesn't go off, the second alarm is triggered and the logger advances both alarms to the next logging interval. It also records the time at which the failure of the first alarm took place. Thus far, I have never seen this in use outside of test cases in which we force the first alarm to fail.

Interfacing with the real-time-clock is performed through the **DS3231 library**, which is a combination of RTClib (a standard real-time clock library by J.-C. Wippler) and code to run the DS3231 written by E. Ayars and edited by A. Wickert.

All times are represented as a **UNIX Epoch**, seconds since January 1st, 1970. This is a very widespread format with plenty of converters into conventional date/time, and solves the problems of (a) storage of date/time data into a complex data type on a small system, and (b) daylight savings time and leap years.

SD card

Utilities to read and write to and from the SD card are managed via the **SdFat** library, by Bill Greiman. The ALog will write to the following files, in rough order of how common their use is:

Always included:

- ∗∗[DataFileCustomName].txt∗∗: A data file, with the name chosen by the user; for safety's sake, I always go with 8.3 format, but this is more old habit.
- **header.txt**: A header file with the names of each column listed. A new timestamped row is added each time the logger is rebooted, to ensure that the data in the data file may always be matched to their appropriate real-world significance.

Sometimes included:

- **bucket_tips.txt** records the times at which a tipping-bucket rain gauge's bucket tips; more tips over a time interval means heavier rain!
- **Camera.txt** records the times at which an attached camera that may lack its own clock takes photos or video
- **Oversample.txt** contains every individual analog reading that is combined into an oversampled resultant measurement, if the "debug" flag is set to `True`
- **Alarm_miss.txt**: Any time the first alarm has not gone off and the backup alarm has had to save the day, a time stamp is written to this file.

Power to the SD card is cut between logging intervals to conserve energy. This is one reason why we do not use the built-in Arduino SD library, which cannot handle losing power to the SD card (or at least could not back when I first tried, in 2011!)

Watchdog timer

The watchdog timer will automatically reboot the ALog data logger if it stalls out while logging for more than 8 seconds. This can help to recover from any software errors, as well as any unforeseen circumstances arising in the field (e.g., jostled while logging causing a disruption of some connections). This can really save the day!

LED messages

These messages are flashed on the large red LED. Those in bold are those that you may see most commonly.

- Syncopated: **daaaa-da-daaaa-da-daaaa-da**: Clock has reset to the year 2000! Please set the clock.
- **LONG-short-short**: All is good! Starting to log.
- 5 quick flashes: Missed first alarm; caught by backup alarm. Rebooting logger.
- **20 quick flashes**: SD card failed, or (more likely) is not present. This also happens if it is not seated properly; reseat the card and try again.
- 50 quick flashes: you have tried to reassign a pin critical to the ALog to another function; this will most likely break the system. Check your code and re-upload.
- 100 quick flashes: your Arduino model is not recognized by the ALog library!

EEPROM: Serial number and calibration values for voltage regulators

The EEPROM of an ALog data logger or other Arduino-based device is the memory that remains unchanged even if it is reprogrammed. It holds important **identifying and calibration information** about the ALog data logger.

- The **serial number** of the data logger is a 16-bit integer held in the first two bytes of the EEPROM (bytes 0 and 1)
- The **measured voltage of the 3.3V regulator** on the ALog BottleLogger, if recorded, is stored in bytes 2-5 of the EEPROM.
- The **measured voltage of the 5V charge pump** on the ALog BottleLogger, if recorded, is stored in bytes 6-9 of the EEPROM.

These can be read by the functions:

- get_serial_number()
- get_3V3_measured_voltage()
- get_5V_measured_voltage()

You can use these functions with sensors by passing the results from a call to `get_3V3_measured_voltage()` or `get_5V_measured_voltage()` to functions as `Vsupply` or `Vref`, these being the supply an reference voltages for the sensors, respectively.

Sensor functions (mostly "public" functions, exposed to the user)

The full index of the sensor functions, along with ways to use them, is here:\ http://northernwidget.↩ github.io/ALog/classALog.html

See the thermistor example (below) and the information in the "Basic Reference" (above) for a more in-depth description of how to create and use sensor functions.

Thermistor example

Thus far, we have explored many different aspects of the ALog bottle logger. Now, let's bring many of these together in an example of source code for the ALog. In this example, we will measure a temperature-sensitive resistor, or "thermistor", using the analog measurement capabilities of the ALog. This is one of the most basic measurements that one can make with the ALog, but also one of the most important and most commonly employed.

```
#include "ALog.h"

ALog alog;

// Note: Serial baud rate is set to 38400 bps

// USER-ENTERED VARIABLES //
char* dataLoggerName = "Thermistor Logger"; // Name of logger; displayed in Serial communications
char* fileName = "TempLog1.txt"; // Name of file for logged data: 8.3 format (e.g,
                           // ABCDEFGH.TXT); <= 8 chars before ".txt" is OK

//Setup logging interval here, may set up more than one variable.
//Minimum interval = 1 sec, maximum interval is 23 hours, 59 minutes, 59 seconds
//0 for all means that the logger will not sleep
int Log_Interval_Seconds = 10; //Valid range is 0-59 seconds
int Log_Interval_Minutes = 0; //Valid range is 0-59 minutes
int Log_Interval_Hours = 0; //Valid range is 0-23 hours
// external_interrupt is true for a tipping bucket rain gauge
bool external_interrupt = false;

void setup(){
  alog.initialize(dataLoggerName, fileName,
    Log_Interval_Hours, Log_Interval_Minutes, Log_Interval_Seconds,
    external_interrupt);

  // If you are using a standard Arduino board (i.e. not a full ALog data
  // logger) and are not using the Arduino shield, you will have to set the
  // proper pins for the indicator LED (defaults to 9) and the SD card, RTC
  // power, and sensor or other external device power (default to -1).
  // The "-1" default ensures that no pins are affected by commands to switch
  // power to the SD card, RTC power, or sensor power. This is because the
  // typical use case, a standard Arduino Uno, uses the Arduino's built-in
  // 3V3 regulator to power to run the RTC and SD card (as we do with the ALog
  // shield).
  // These can also be set to the same value if they are needed but the same
  // voltage regulator serves multiple purposes.
  // Replace "_pin" with your desired pin number, and uncomment the relevant
  // following line(s).
  // set_LEDpin(_pin);
  // set_SDpowerPin(_pin);
  // set_RTCpowerPin(_pin);
  // set_SensorPowerPin(_pin);

  alog.setupLogger();
}

void loop(){
  // ****************************************************

  alog.goToSleep_if_needed(); // Send logger to sleep
  alog.startLogging(); // Power up all systems, check WDT, reset alarms,
                       // and open data file(s) in write mode

  // ************ DO NOT EDIT ABOVE THIS LINE ************
```

```
  // READ SENSORS; GATHER/PROCESS DATA: EDIT THIS PART //

  alog.sensorPowerOn();

  // Turn on external power (3.3V and 5V in the case of the ALog BottleLogger)
  // for sensors and any other devices.
  // Place commands for all sensors that require this between
  // SensorPowerOn() and SensorPowerOff().
  // If you have no sensors that require power, you should comment out the
  // SensorPowerOn() and SensorPowerOff() commands.

  // CanTherm small bead
  //logger.thermistorB(10000, 3950, 10000, 25, 0);
  // CanTherm epoxy bead: CWF1B103F3380
  logger.thermistorB(10000, 3380, 10000, 25, 0);

  alog.sensorPowerOff();

  // !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!! //
  // INSERT COMMANDS TO READ SENSORS THAT DO NOT REQUIRE ALOG-SUPPLIED POWER HERE! //
  // !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!! //


  // NOTE: THE BUFFER SIZE IS 512 BYTES;
  // run "alog.bufferWrite()" between your commands in this section
  // if you think you are approaching this limit.
  // Otherwise, the buffer will overflow and I'm not sure what will happen!
  // alog.bufferWrite()

  // ************ DO NOT EDIT BELOW THIS LINE ************

  // Wrap up files, turn off SD card, and go back to sleep
  alog.endLogging();

  // *****************************************************
}
```

It's time to break this down into its component steps:

```
#include "ALog.h"

ALog alog;
```

The `#include` call tells the sketch to use the ALog library.

The second line of source code creates an instance of the ALog object, called "alog". We will use "alog" as our portal into all of the tools that the ALog library offers.

```
// Note: Serial baud rate is set to 38400 bps
```

Self-explanatory!

```
// USER-ENTERED VARIABLES //
char* dataLoggerName = "Thermistor Logger"; // Name of logger; displayed in Serial communications
char* fileName = "TempLog1.txt"; // Name of file for logged data: 8.3 format (e.g,
                          // ABCDEFGH.TXT); <= 8 chars before ".txt" is OK

//Setup logging interval here, may set up more than one variable.
//Minimum interval = 1 sec, maximum interval is 23 hours, 59 minutes, 59 seconds
//0 for all means that the logger will not sleep
int Log_Interval_Seconds = 10; //Valid range is 0-59 seconds
int Log_Interval_Minutes = 0; //Valid range is 0-59 minutes
int Log_Interval_Hours = 0; //Valid range is 0-23 hours
// external_interrupt is true for a tipping bucket rain gauge
bool external_interrupt = false;
```

`dataLoggerName` is an identifier for the logger. I typically make it relate to the field site, the task of the logger, and a code number.

`fileName` is the name of the main data file to be logged to the SD card. (See SD card section, above, for information on all of the files that may be written to the SD card.) This is noted to be in 8.3 format, but doesn't strictly have to be following updates to the `SdFat` library.

`Log_Interval_Seconds`, `Log_Interval_Minutes`, and `Log_Interval_Hours` all determine how often the logger will record data. Data recording is set to always occur on hours/minutes/seconds that are synchronous across all devices and are set to be referenced as much as possible to a the start of an hour/day/minute (more details in source code).

`external_interrupt` is true if a device that triggers an instant response is attached, and false if it is not. As noted, this is most commonly a tipping-bucket rain gauge, but it could really be any sensor. The appropriate sensor functions will define the response to the interrupt; the simplest case (rain gauge) is that a time-stamp is recorded to a specific file.

```
void setup(){
  alog.initialize(dataLoggerName, fileName,
    Log_Interval_Hours, Log_Interval_Minutes, Log_Interval_Seconds,
    external_interrupt);
```

This is the start of the `setup()` step, which is run once when the logger turns on. Here, the variables from the previous section are passed to the "alog" object instance, which then creates the appropriate variables and their derivatives within the object. This initializes the object and prepares it for logging.

```
// If you are using a standard Arduino board (i.e. not a full ALog data logger)
// and are not using the Arduino shield, you will have to set the proper pins for
// the indicator LED (defaults to 9) and the SD card and RTC power (default to -1
// if you are not programming an ALog data logger in order
// to be inactive in the case that you are using the Arduino's 3V3 regulator to
// power to run the RTC and SD card (as we do with the ALog shield); set these to the same
// value if there is just one switch for both of these).
// Replace "_pin" with your desired pin number, and uncomment the relevant line(s).
// set_LEDpin(_pin);
// set_SDpowerPin(_pin);
// set_RTCpowerPin(_pin);
```

As is written in the comments, set these values for the indicator LED, SDpowerPin, and RTCpowerPin. The latter two are typically unset on non-ALog Arduino boards, and are equal to each other (but already set within the ALog library) for ALog boards.

```
  alog.setupLogger();
}
```

At this point, the logger has all the information it needs from you, and completes its internal setup.

```
void loop(){
  // *************************************************

  alog.goToSleep_if_needed(); // Send logger to sleep
  alog.startLogging(); // Power up all systems, check WDT, reset alarms,
                       // and open data file(s) in write mode

  // ************ DO NOT EDIT ABOVE THIS LINE ************
```

This is the start of the loop, which repeats infinitely unless the ALog data logger is shut off or fails.

Here, the comments are self-explanatory, mostly; a couple extra explanations:

- "Sleep" is a low-power mode; it will exit this (and complete this function) when triggered by the clock, a press of the "LOG" button, or an user-defined external interrupt.
- WDT = watchdog timer; this causes the logger to reboot if it hangs for >8 seconds, and therefore allows it to recover from failures.

```
// READ SENSORS; GATHER/PROCESS DATA: EDIT THIS PART //

alog.sensorPowerOn();

// Turn on external power (3.3V and 5V in the case of the ALog BottleLogger)
// for sensors and any other devices.
// Place commands for all sensors that require this between
// SensorPowerOn() and SensorPowerOff().
// If you have no sensors that require power, you should comment out the
// SensorPowerOn() and SensorPowerOff() commands.

// CanTherm small bead
//logger.thermistorB(10000, 3950, 10000, 25, 0);
// CanTherm epoxy bead: CWF1B103F3380
logger.thermistorB(10000, 3380, 10000, 25, 0);

alog.sensorPowerOff();
```

This is it, where the magic happens!

`alog.sensorPowerOn()` and `alog.sensorPowerOff()` turn the voltage regulators that power external sensors on and off, respectively – just like it says in the comments!

`logger.thermistorB(...)` is the command. The parameters passed, in order, are:

- Factory standard resistance of thermistor [ohms]
- Factory standard b-value (or beta-value) of thermistor [ohms]
- Reference resistor resistance [ohms]
- Standard temperature to which factory standard resistance corresponds [ohms]
- Analog pin number (this could equally be A1 or 14 for the ALog BottleLogger)

Do you want to know more about these parameters? Look up the function-specific help, which is below if you are looking at the PDF version of the help (downloadable from http://northernwidget.github.io/ALog/ALog↩ReferenceManual.pdf) and available online at http://northernwidget.github.io/ALog/class↩ALog.html.

```
// !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!! //
// INSERT COMMANDS TO READ SENSORS THAT DO NOT REQUIRE ALOG-SUPPLIED POWER HERE! //
// !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!! //
```

We don't have any of these here, but examples would be sensors with their own power-supplies, and/or (not consistent with the comment above) sensors that are powered directly via the ALog BottleLogger's VCC. This is not included in the comment because it is generally not how we run sensors – we like to be able to switch them off to conserve power!

```
// NOTE: THE BUFFER SIZE IS 512 BYTES;
// run "alog.bufferWrite()" between your commands in this section
// if you think you are approaching this limit.
// Otherwise, the buffer will overflow and I'm not sure what will happen!
// alog.bufferWrite()
```

As it says: if you think you're in danger of overflowing the buffer, run this command! You'll need 512 characters in your line for this to happen though, and I've never approached this in my work. But hey, someone might!

```
  // ************ DO NOT EDIT BELOW THIS LINE ************

  // Wrap up files, turn off SD card, and go back to sleep
  alog.endLogging();

  // ***************************************************
}
```

As it says, the `alog.endLogging()` command finalizes everything before sending the logger back to sleep. And that's really all there is to it! Your first ALog Arduino sketch in a nutshell.

And as always, if anything here is unclear, please contact us and we'll try our best to fix it – specific advice appreciated!

Setting the real-time clock

Graphical interface

Coming soon!

Command-line

The ALog's clock is a key component – without it, all the data are sitting in an unreferenced time frame. This is why we have chosen the DS3231 real-time clock, which has the highest accuracy and stability of any commonly-available real-time clock that does not incorporate an absolute time reference (e.g., with GPS). GPS would be great for some circumstances, and we're developing a logger with it, but this would not be the extreme low-power device that the ALog BottleLogger is.

Enough with the background – how do you set the clock?

You will need the **ALog library** and the **DS3231 library**, both of which you should already have at this point, as well as the **ALogTalk** repository of Python scripts at `github.com/NorthernWidget/ALogTalk`. In order to run the Python scripts, you need either **Python 2** (including **pyserial**) or for us to have made an executable version for your platform.

Integrated into the ALog code

When you upload the ALog code to your ALog data logger or Arduino board, it includes code to set the clock via serial communications. Plug in your data logger, run **ALogClockSet.py** (which is in the **ALogTalk** repository), and it should work – if it doesn't, hit the reset button.

How do you run **ALogClockSet.py**?

All operating systems: Open a terminal, navigate to the directory that holds **ALogClockSet.py** (the directory should be called **ALogTalk**, and type:

```
python ALogClockSet.py
```

Don't have Python? You probably will have figured it out when that failed Here are your options:

1. Install Python 2.X; I recommend getting the **anaconda** distribution
2. Download an executable version of **ALogClockSet.py**; email us if you can't find this (it's not always up-to-date due to the overhead required for making executables)

After uploading a separate sketch

Open the "ALog_DS3231_set_echo" example (**File (Arduino on Mac)** > **Examples** > **DS3231** > **ALog_DS3231↩ _set_echo**) and upload it to the board.

Open a terminal, navigate to the **ALogTalk** directory, and type:

```
python SimpleClockSet.py
```

Connecting sensors (and more) to the ALog

\

ALog BottleLogger v.2.2.0
User's layout

Legend

- Ground (GND; 0V)
- Power (VCC)
  - 3.6 - 4.7 V with 3 primary Alkaline cell batteries via jack/plug
  - [OR} 5V powered via USB
- Switched voltage supply: 3.3V or 5V*
  - *5V charge pump provides up to 5.2V until current draw hits 5-10 mA
- Digitial inputs and outputs (I/O):
  - In-system programmer access: burn bootloader (or programs directly)
  - MOSI, MISO, SCK: SPI bus (also used for SD card)
  - D9, D6: PWM-enabled digital I/O pins
  - D3-INT1: External interrupt can be read at this digital I/O pin
  - RX, TX: UART (also connected to USB comms via 1kΩ in-series resistors)
  - SDA, SCL: I2C bus operating at 3.3V (also connected to RTC)
  - USB: Communication with logger via USB-Serial converter and UART
- Analog inputs, referenced to 3.3V
- Reference resistor headers for voltage dividers

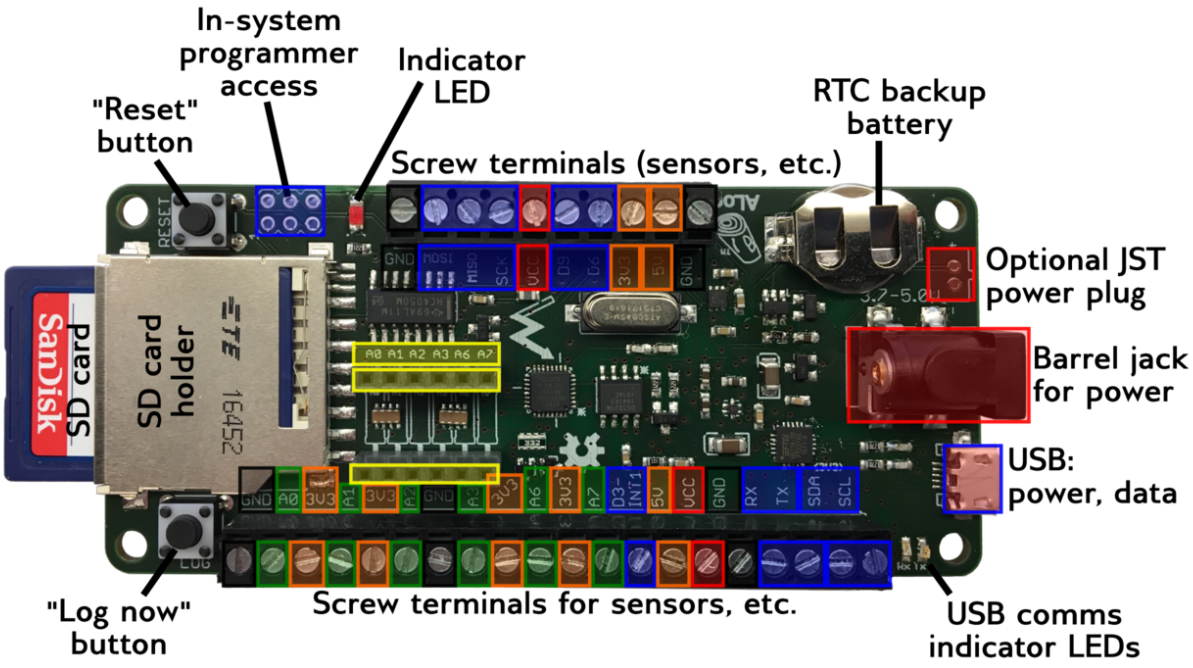


Figure 1.20 Guide

User's layout and connection guide

[PDF of User's Layout guide]\(backup link)

Pin definitions

This list includes only those pins on the ALog BottleLogger with which the user can connect.

Pin number (Arduino) Label on ALog BottleLogger Capabilities

0 Rx UART Rx∗ 1 Tx UART Tx∗ 3 D3/INT1 External interrupt; digital I/O 6 D6 PWM∗∗; digital I/O 9 D9 PWM∗∗; digital I/O 11 MOSI SPI MOSI 12 MISO SPI MISO 13 SCK SPI SCK A0 (14) A0 Analog input (10-bit); digital I/O A1 (15) A1 Analog input (10-bit); digital I/O A2 (16) A2 Analog input (10-bit); digital I/O A3 (17) A3 Analog input (10-bit); digital I/O A4 (18) / SDA SDA I2C SDA A5 (19) / SCL SCL I2C SCL A6 (20) A6 Analog input (10-bit); digital I/O A7 (21) A7 Analog input (10-bit); digital I/O

∗The UART is connected to the USB-Serial converter; using it with sensors is possible, but requires caution when designing new UART sensor interfaces to avoid interference with communications with the computer.

∗∗PWM = "Pulse-width modulation"

This table does not include pin capabilities that are technically possible but not recommended. These include:

- Rx and Tx pins (UART), SPI pins, and I2C pins can all be used as standard digital I/O pins; this would cause problems communicating with the computer, communicating with the SD card (and sensors), and communicating with the clock (and sensors), respectively.
- I2C pins (A4 and A5) are not recommended to be used as analog input pins, though technically they are; this will cause problems with communication with the real-time clock and any other I2C device

Additional **ALog** ports

Label on ALog BottleLogger Connected to...

GND Ground (Earth / 0V / battery negative (-) terminal) 3V3 3.3V ±1% precision low-dropout voltage regulator 5V 5V charge pump; 5V is nominal: for tiny loads, it can go up to 5.2V VCC "Voltage of the common connector" (Power / battery (+) terminal)

How to attach wires to screw terminals

Tools/materials needed:

- Screwdriver, slotted. We recommend a 0.4 x 2 mm blade. Good options are this one and its ESD-safe version.
- ALog data logger or equivalent
- A sensor!

Also helpful:

- Common sense
- Caution

To be safe, disconnect power from the logger before attaching or detaching any wires. For full disclosure, though, we do this with power attached all the time – this just opens the possibility of something going wrong, and requires steady hands.

Using the screwdriver, open the screw terminal (lefty loosey), stick the end of the wire into it, and then close the screw terminal until you feel significant resistance (righty tighty!). The goal is that the wire can't come out under reasonable force.

![Attaching a wire (photo R. Schulz)](https://github.com/NorthernWidget/ALog/raw/master/doc/figures/AttachWireScrew.png "Attaching a wire (photo R. Schulz)")\

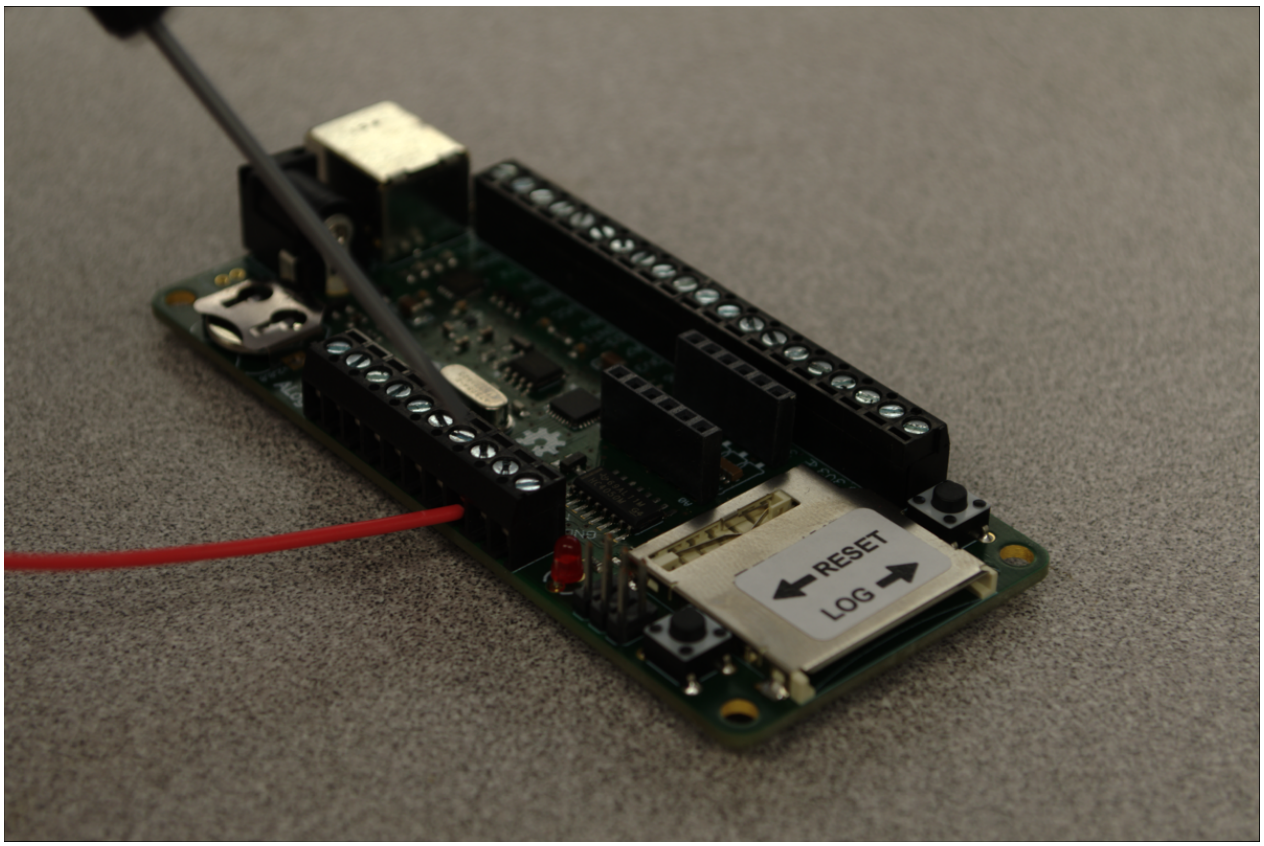


Figure 1.21 Attaching wire

Reference resistors (what are they good for?)

Two parallel rows of headers are designed to hold through-hole resistors that span one row to the other. Each of these is numbered to correspond to a particular analog port. Why are these here?

The analog channels on the ALog (or any electronic device) read voltage. But a lot of sensors return resistance. So how do we change that resistance into a voltage?

The key is a little bit of circuit algebra for something called the "Voltage divider":

\

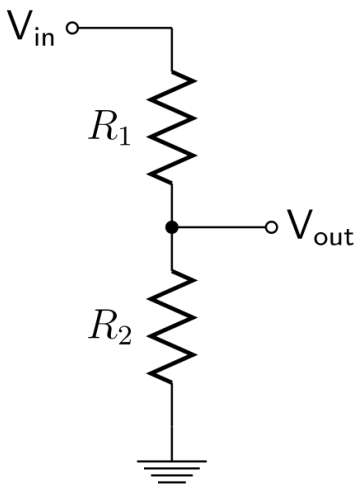


Figure 1.22 Voltage divider

For the ALog, the sensor typically lies between 3V3 (a power source, Vin on the above diagram) and an analog pin (Vout on the above diagram). Applying Ohm's law gives you the following equation that one can use to solve for the value of R1 if you know R2.

\

$$R_1 = R_2 \left(\frac{V_{\text{in}}}{V_{\text{out}}} - 1 \right)$$

And this is where the reference resistor comes in: this is the R2, connected between Vout and GND.

It is possible, and necessary with a few sensors, to flip the structure around; in this case, a reference resistor is not attached to the header on the board, but is instead attached to the screw terminals for a power source and the appropriate analog port. Such a reversed order may require a flag to be set in the code.

And that's it: **the reference resistor is a standard** required to measure another unknown resistance, and it is implemented in a circuit that converts the unknown resistance into a voltage.

For more information, see `Wikipedia: Voltage divider`

The ISP (or ICSP) header and the bootloader

This is the last section in the connection guide because it is the one that – hopefully – you will never have to use!

Place your ALog BottleLogger in front of you with the SD card to the right. Look at the upper left corner. Do you see those six holes? Those are the ones that we are talking about. They allow the user to burn programs directly onto the logger. **This is called the "In-System Programmer" or "ISP" header.**

∗(If you are using a standard Arduino, there should be an ISP header as well; if you are using an ARM-based chip, there is a bigger header with smaller pin spacings.)∗

When your ALog (or Arduino) arrives, it has a bootloader on it – a small program that allows you to upload code via the USB connection. Useful! But if somehow something happens to the bootloader, you will have to burn a new one. You will need an in-system programmer: AVR ISP, another Arduino, the Adafruit USBtinyISP, etc... there are many possibilities! If you have an ARM-based chip, you will need an ATmel AVR Dragon or other more advanced programming chip.

Then, remove the SD card. The ISP on any standard Arduino uses the SPI interface. The SD card, which also uses the SPI interface, may interfere with bootloader burning.

After this, go to the **Tools** menu. Make sure that you have selected the proper board (**Tools** > **Board**) and programmer (**Tools** > **Programmer: "\[NAME OF YOUR CURRENTLY-SELECTED PROGRAMMER\]"**). Then plug the ISP of your choice into the ISP header: simply holding a 6-pin header at an angle in the holes so the metal makes contact is enough for the ALog BottleLogger, but you may solder it if you choose. Once you have a good connection, go to **Tools** > **Burn Bootloader** and wait for the programmer to tell you that it has successfully completed the task.

If it doesn't work, the most common reason is because you have inserted the ISP backwards. Flip the ISP header around and try again.

Field deployment

SD card

Insert the SD card carefully, and make sure that it is seated! The logger will reset constantly if it is not and it is v2.2.↩
0+. Otherwise, it is important to hit the "RESET" button and make sure that the LOOOOONG-short-short flash pattern occurs. This means that the logger is ready to go. Otherwise, no data will be recorded!

![Inserting SD card (photo R. Schulz)](https://github.com/NorthernWidget/ALog/raw/master/doc/figures/↩
InsertSD.png "Inserting SD card (photo R. Schulz)")\

Figure 1.23 Inserting SD card

Power

Barrel jack

In the field, the main way to power the ALog data logger is by plugging in a barrel jack connected to a battery pack. We typically use barrel jacks with screw terminals to attach to generic battery packs (D or AA) that have wire leads; we sometimes use barrel jacks that are already part of the cable.

![Plugging in the barrel jack (photo R. Schulz)](https://github.com/NorthernWidget/ALog/raw/master/doc/figures
BarrelJack.png "Plugging in the barrel jack (photo R. Schulz)")\

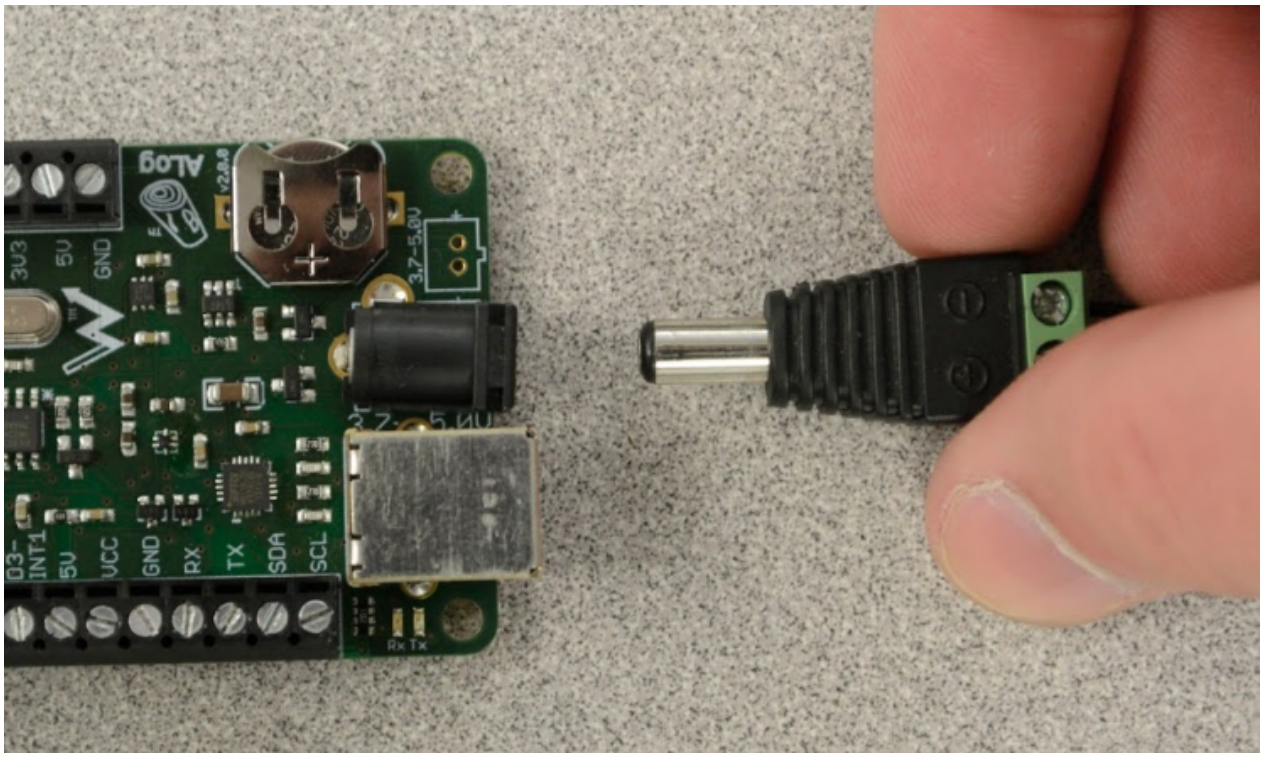


Figure 1.24 Barrel jack power

JST connector

Next to the barrel jack is a solder pad for a JST connector. This may be populated (i.e., you may solder a JST connector to the board here) if needed to attach power supplies.

Power sources

We typically use 3xD or 3xAA primary cell batteries, as they are light, inexpensive, reliable, and available around the world.

Solar charging system development is ongoing, and once complete, will allow long-term power supply and a higher rate of logger power consumption.

Housing and waterproofing

![Housing, logger, AA battery box, and cable gland (photo R. Schulz)](https://github.com/NorthernWidget/ALog/raw/master/doc/figures/HousingSmall.png "Housing, logger, AA battery box, and cable gland (photo R. Schulz)")\

Figure 1.25 Housing

ALog in a box. Note that here, the ALog is powered through its VCC and GND screw terminals; this demonstrates a non-standard but acceptable way of supplying power to the logger, and one that works well in a small enclosure like this one.

Housing details

- Manufacturer: Bud Industries
- Ratings: IP66, NEMA 1, 2, 4, 4X, 12, 13
- Description: Grey ABS plastic with a hinged door lid. Mounting hardware included.
- Preferred sizes:
  - Small (pictured above): Max exterior dimensions 150mm H X 100mm W X 70mm H, Wall thickness 3mm, Weight: 318g
  - Large: Max exterior dimensions 270mm H X 100mm W X 70mm H, Wall thickness 3mm, Weight: 363g
- Metal "feet" can swiveled and work with U-bolts, hose clamps, screws, cable ties, etc.

Battery pack

- Box with lid, connected to housing with hook-and-loop.

ALog

- Fixed in place with hook-and-loop.

Cable gland

- Nylon cable gland: NBR hermetic seal. Protection degree: IP68, 5 Bar, IP69K, UL f1. Working temperature: -40℃ to 100℃ in static state Instantaneous heat resistance up to 120℃ -20℃ to 80℃ in dynamic state. cable diameter range 0.11-0.26"

Vent (if needed)

- GORE VE80510 adhesive vent for pressure equalization. Vent offers reliable protection against water, salts, corrosive liquids and particulate with hydrophobic and oleophobic characteristics. This vent is made of GORE-↩ TEX fabric to keep water out while permitting the passage of air.

Deployment

The last step is deployment! You have to think about how you will mount the logger box:

- Bury it?
- Attach it to a post?
- Place it on the ground?

\

Figure 1.26 Chimborazo

It also is important to consider how you will attach it: do you need:

- Zip-ties
- Hose clamps
- U-bolts
- Electrical tape
- Bailing wire
- Fence posts (and fence post driver?)
- Desiccant packets
- Tools
- Specific tools for your installation?

There are so many ways to mount the loggers in the field that we can't cover all usage scenarios. But here are some important points:

1. It is really important to think through your deployment! Use common sense, test locally, and ask for advice!
2. Label the SD card, box, and the logger with an identifying code or name for the station.
3. Ensure that your code that you have uploaded includes a unique filename and logger name.
4. If you have cable glands, point them down if at all possible to prevent water from entering.
5. Desiccant packs are your friends; make sure that you don't use the kind that fail catastrophically upon reaching 100% moisture content by turning into a liquid! (We had this happen in Ecuador...)
6. Assess the greatest challenges. If weather, place your logger in a safe place (typically high and dry). If humans, hide it. (Bury it, even! But take a good GPS location and make note of landmarks.)
7. Record the logger's name, its sensors, its serial number, and its GPS coordinates.
8. Small rodents love chewing cables for reasons that are their own (but might have something to do with the taste of the insulation). Bury cables, keep them high, and/or surround them with conduit. (I've seen plastic conduit chewed through once as well...)
9. The last thing before you leave an ALog installation should always be to push the "RESET" button and ensure that you see "LOOOONG short short" from the red LED (and only "LOOOONG short short")!

So long, and good luck!

Chapter 2

Class Documentation

2.1 ALog Class Reference

Public Member Functions

- ALog ()

  ALog library for the Arduino-based data loggers.
- void initialize (char *_logger_name, char *_datafilename, int _hourInterval, int _minInterval, int _secInterval, bool _ext_int=false, bool _LOG_ALL_SENSORS_ON_BUCKET_TIP=false)
- void setupLogger ()
- void sleep ()
- void goToSleep_if_needed ()
- void startLogging ()
- void endLogging ()
- void startAnalog ()
- void endAnalog ()
- void sensorPowerOn ()
- void sensorPowerOff ()
- bool get_use_sleep_mode ()
- void set_LEDpin (int8_t _pin)
- void set_SDpowerPin (int8_t _pin)
- void set_RTCpowerPin (int8_t _pin)
- void set_SensorPowerPin (int8_t _pin)
- uint16_t get_serial_number ()
- float get_3V3_measured_voltage ()
- float get_5V_measured_voltage ()
- float readPin (uint8_t pin)
- float readPinOversample (uint8_t pin, uint8_t adc_bits)
- float analogReadOversample (uint8_t pin, uint8_t adc_bits=10, uint8_t nsamples=1, bool debug=false)
- float thermistorB (float R0, float B, float Rref, float T0degC, uint8_t thermPin, uint8_t ADC_resolution_nbits=14, bool Rref_on_GND_side=true, bool oversample_debug=false, bool record_results=true)
- void ultrasonicMB_analog_1cm (uint8_t nping, uint8_t EX, uint8_t sonicPin, bool writeAll)
- float maxbotixHRXL_WR_Serial (uint8_t Ex, uint8_t npings, bool writeAll, int maxRange, bool RS232=false)

- void maxbotixHRXL_WR_analog (uint8_t nping=10, uint8_t sonicPin=A0, uint8_t EX=99, bool writeAll=true, uint8_t ADC_resolution_nbits=10)
- void Decagon5TE (uint8_t excitPin, uint8_t dataPin)
- void DecagonGS1 (uint8_t pin, float Vref, uint8_t ADC_resolution_nbits=14)
- void vdivR (uint8_t pin, float Rref, uint8_t ADC_resolution_nbits=10, bool Rref_on_GND_side=true)
- void linearPotentiometer (uint8_t linpotPin, float Rref, float slope, char *_distance_units, float intercept=0, uint8_t ADC_resolution_nbits=14, bool Rref_on_GND_side=true)
- void HTM2500LF_humidity_temperature (uint8_t humidPin, uint8_t thermPin, float Rref_therm, uint8_t ADC_resolution_nbits=14)
- void HM1500LF_humidity_with_external_temperature (uint8_t humidPin, float R0_therm, float B_therm, float Rref_therm, float T0degC_therm, uint8_t thermPin_therm, uint8_t ADC_resolution_nbits=14)
- void Inclinometer_SCA100T_D02_analog_Tcorr (uint8_t xPin, uint8_t yPin, float Vref, float Vsupply, float R0_therm, float B_therm, float Rref_therm, float T0degC_therm, uint8_t thermPin_therm, uint8_t ADC_resolution_nbits=14)
- void Anemometer_reed_switch (uint8_t interrupt_pin_number, unsigned long reading_duration_milliseconds, float meters_per_second_per_rotation)
- void Wind_Vane_Inspeed (uint8_t vanePin)
- void Pyranometer (uint8_t analogPin, float raw_mV_per_W_per_m2, float gain, float V_ref, uint8_t ADC_resolution_nbits=14)
- void Barometer_BMP180 ()
- void _sensor_function_template (uint8_t pin, float param1, float param2, uint8_t ADC_bits=14, bool flag=false)
- void HackHD (int control_pin, bool want_camera_on)
- float Honeywell_HSC_analog (int pin, float Vsupply, float Vref, float Pmin, float Pmax, int TransferFunction_number, int units, uint8_t ADC_resolution_nbits=14)

2.1.1 Constructor & Destructor Documentation

2.1.1.1 ALog()

```
ALog::ALog ( )
```

ALog library for the Arduino-based data loggers.

ALog data logger library: methods to:

- Initialize the data logger
- Sleep and wake
- Interact with the real-time clock (RTC)
- Write data to the SD card
- Manage power
- Interact with a range of sensors

All help documentation here assumes you have created an instance of the "ALog" class.

```
ALog alog;
```

2.1.2 Member Function Documentation

2.1.2.1 _sensor_function_template()

```
void ALog::_sensor_function_template (
            uint8_t pin,
            float param1,
            float param2,
            uint8_t ADC_bits = 14,
            bool flag = false )
```

Function to help lay out a new sensor interface. This need not be "void": it may return a value as well.

Details about sensor go here

Parameters

pin	You often need to specify interface pins
param1	A variable for the sensor or to interpret its value
param2	A variable for the sensor or to interpret its value
ADC_bits	You often need to specify how much the analog-to-digital converter should be oversampled; this can range from 10 (no oversampling) to 16 (maximum possible oversampling before certainty in the oversampling method drops)
flag	Something that helps to set an option

Example (made up):

```
alog.Example(A2, 1021.3, 15.2, True);
```

2.1.2.2 analogReadOversample()

```
float ALog::analogReadOversample (
            uint8_t pin,
            uint8_t adc_bits = 10,
            uint8_t nsamples = 1,
            bool debug = false )
```

Higher analog resolution through oversampling

This function incorporates oversampling to extend the ADC precision past ten bits by taking more readings and statistically combing them.

Returns a floating point number between 0 and 1023 in order to be intechangable with the Arduino core AnalogRead() function

It is often used within other sensor functinons to increase measurement precision.

Parameters

pin	is the analog pin number
adc_bits	is the reading precision in bits (2^adc_bits). The ATMega328 (Arduino Uno and ALog BottleLogger core chip) has a base ADC precision of 10 bits (returns values of 0-1023) A reasonable maximum precision gain is (base_value_bits)+6, so 16 bits is a reasonable maximum precision for the ALog BottleLogger.
nsamples	is the number of times you want to poll the particular sensor and write the output to file.
debug	is a flag that, if true, will write all of the values read during the oversampling to "Oversample.txt".

Example:

```
// 12-bit measurement of Pin 2
// Leaves nsamples at its default value of 1 (single reading of sensor)
alog.analogReadOversample(2, 12);
```

Readings that require more bits of precision will take longer.

For analog measurements that do not require more than 10 bits of precision, use alog.readpin(int pin) or the standard Arduino "AnalogRead" function.

Based on eRCaGuy_NewAnalogRead::takeSamples(uint8_t analogPin)

Example:

```
// Take a single sample at 14-bit resolution and store it as "myReading"
myReading = alog.analogReadOversample(A3, 14, 1);
```

2.1.2.3 Anemometer_reed_switch()

```
void ALog::Anemometer_reed_switch (
              uint8_t interrupt_pin_number,
              unsigned long reading_duration_milliseconds,
              float meters_per_second_per_rotation )
```

Anemometer that flips a reed switch each time it spins.

Parameters

interrupt_pin_number	is the digital pin number corresponding to the appropriate interrupt; it uses the Arduino digitalPinToInterrupt(n_pin) function to properly attach the interrupt. On the ALog BottleLogger, this number will always be **3**.
reading_duration_milliseconds	How long will you count revolutions? Shorter durations save power, longer durations increase accuracy; very long durations will produce long-term averages. Typical values are a few seconds.
meters_per_second_per_rotation	Conversion factor between revolutions and wind speed. For the Inspeed Vortex wind sensor that we have used (http://www.inspeed.↩
	com/anemometers/Vortex_Wind_Sensor.asp), this is: **2.5 mph/Hz = 1.1176 (m/s)/Hz**

This function depends on the global variable **rotation_count**.

Example:

```
// 4-second reading with Inspeed Vortex wind sensor on digital pin 3
// (interrupt 1), returned in meters per second
alog.Anemometer_reed_switch(3, 4000, 1.1176);
```

2.1.2.4 Barometer_BMP180()

```
void ALog::Barometer_BMP180 ( )
```

Read absolute pressure in mbar.

This function reads the absolute pressure in mbar (hPa). BMP180 sensor incorporates on board temperature correction. Uses I2C protocol.

Example:

```
alog.Barometer_BMP180();
```

2.1.2.5 Decagon5TE()

```
void ALog::Decagon5TE (
            uint8_t excitPin,
            uint8_t dataPin )
```

Reads a Decagon Devices 5TE soil moisture probe.

NEEDS TESTING with current ALog version.

Returns Dielectric permittivity [-unitless-], electrical conductivity [dS/m], and temperature [degrees C]. Soil moisture is calculated through postprocessing.

Uses **SoftwareSerial**, and therefore has the potential to go unstable; however, we have a time limit, so this won't crash the logger: it will just keep the logger from recording good data.

Modified from Steve Hicks' code for an LCD reader by Andy Wickert

Parameters

excitPin	activates the probe and powers it
dataPin	receives incoming serial data at 1200 bps

Example:

```
alog.Decagon5TE(7, 8);
```

2.1.2.6 DecagonGS1()

```
void ALog::DecagonGS1 (
            uint8_t pin,
            float Vref,
            uint8_t ADC_resolution_nbits = 14 )
```

Ruggedized Decagon Devices soil moisture sensor

Parameters

pin	Analog pin number
Vref	is the reference voltage of the ADC; on the ALog, this is a precision 3.3V regulator (unless a special unit without this regulator is ordered; the regulator uses significant power)
ADC_resolution_nbits	(10-16 for the ALog BottleLogger) is the number of bits of ADC resolution used (oversampling for >10 bits)

Example:

```
// Using a non-precision Rref that is slightly off
alog.DecagonGS1(A1, 3.27, 14);
```

2.1.2.7 endAnalog()

```
void ALog::endAnalog ( )
```

Turn off power to analog sensors

Actually just turns off 3V3 regulator that could power anything. Keeping this around for backwards compatibility; look to sensorPowerOff() for the preferred version of this.

2.1.2.8 endLogging()

```
void ALog::endLogging ( )
```

Endslogging and returns to sleep

Ends line, turns of SD card, and resets alarm: ready to sleep.

Also runs tipping bucket rain gauge code (function that records time stamp) if one is attached and activated.

IMPORTANT: If the logger is not writing data to the card, and the card is properly inserted, the manually-set delay here may be the problem. We think we have made it long enough, but because it is hard-coded, there could be an unforeseen circumstance in which it is not!

2.1.2.9 get_3V3_measured_voltage()

```
float ALog::get_3V3_measured_voltage ( )
```

Retrieve the ALog's 3.3V regulator's actual measured voltage under load, stored in the EEPROM.

It is stored in bytes 2, 3, 4, and 5 of the EEPROM.

2.1.2.10 get_5V_measured_voltage()

```
float ALog::get_5V_measured_voltage ( )
```

Retrieve the ALog's 5V charge pump's actual measured voltage under load, stored in the EEPROM.

It is stored in bytes 6, 7, 8, and 9 of the EEPROM.

2.1.2.11 get_serial_number()

```
uint16_t ALog::get_serial_number ( )
```

Retrieve the ALog's serial number from EEPROM .

It is stored in bytes 0 and 1 of the EEPROM.

2.1.2.12 get_use_sleep_mode()

```
bool ALog::get_use_sleep_mode ( )
```

Does the logger enter a low-power sleep mode? T/F.

- True if the logger is going to sleep between pases through the data-reading loop.
- False if the logger is looping over its logging step (inside void loop() in the ∗.ino code) continuously without sleeping

2.1.2.13 goToSleep_if_needed()

```
void ALog::goToSleep_if_needed ( )
```

Places logger into sleep mode iff this is being used.

Function is accessible from Arduino sketch; is designed for cases in which an external override may be required.

2.1.2.14 HackHD()

```
void ALog::HackHD (
            int control_pin,
            bool want_camera_on )
```

HackHD camera control function

Control the HackHD camera: this function turns the HackHD on or off and records the time stamp from when the HackHD turns on/off in a file called "camera.txt".

Because this function turns the camera on or off, you have to ensure that you write a mechanism to keep it on for some time in your code. This could be checking the time each time you wake and deciding what to do, for example. In short: this function is a lower-level utility that requires the end-user to write the rest of the camera control sequence themselves.

Parameters

control_pin	is the pin connected to the HackHD on/off switch; Dropping control_pin to GND for 200 ms turns camera on or off.
want_camera_on	is true if you want to turn the camera on, false if you want to turn the camera off.

CAMERA_IS_ON is a global varaible attached to this function that saves the state of the camera; it will be compared to "want_camera_on", such that this function will do nothing if the camera is already on (or off) and you want it on (or off).

Power requirements:

- 0.2 mA quiescent current draw;
- 600 mA while recording

Example (not tested):

```
// Before "setup":
uint32_t t_camera_timeout_start_unixtime;
int timeout_secs = 300;
book camera_on = false;
// ...

// Inside "loop":
// Turn the camera on after some triggering event, and keep it on for as
// long as this condition is met, and for at least 5 minutes afterwards.
//
// >> some code to measure a variable's "distance"
// ...
//
if (distance < 1500){
  alog.HackHD(8, true);
  camera_on = true; // Maybe I can get a global variable from this library
                    // or have HackHD function return the camera state?
  now = RTC.now();
  // Reset the timeout clock
  t_camera_timeout_start_unixtime = now.unixtime();
}
else if(camera_on){
  now = RTC.now();
  // If timed out, turn it off.
  if ((t_camera_timeout_start_unixtime - now.unixtime()) > timeout_secs){
    alog.HackHD(8, false);
    camera_on = false;
  }
}
```

This example could be used to capture flow during a flash flood. See:

- Website: http://instaar.colorado.edu/~wickert/atvis/
- AGU poster: https://www.researchgate.net/publication/241478936_The_Automatically_Triggered_Video_or_Imaging_Station_ATVIS_An_Inexpensive_Way_to_Catch_Geomorphic_Events_on_Camera

2.1.2.15 HM1500LF_humidity_with_external_temperature()

```
void ALog::HM1500LF_humidity_with_external_temperature (
             uint8_t humidPin,
             float R0_therm,
             float B_therm,
             float Rref_therm,
             float T0degC_therm,
             uint8_t thermPin_therm,
             uint8_t ADC_resolution_nbits = 14 )
```

HM1500LF Relative humidity sensor with external temperature correction

This function measures the relative humidity of using a HTM1500 relative humidity sensor and an external thermistor. The relative humidity and temperature are measured using an oversampling method. Results are displayed on the serial monitor and saved onto the SD card to four decimal places. Temperature and relative humidity are recorded.

Parameters

humidPin	is the analog pin connected to the humidity output voltage of the module.
R0_therm	is the resistance of the thermistor at the known temperature.
B_therm	is the B- or - parameter of the thermistor.
Rref_therm	is the resistance of the corresponding reference resistor for that analog pin.
T0degC_therm	is a thermistor calibration.
thermPin_therm	is the analog pin connected to the tempurature output voltage of the module.
ADC_resolution_nbits	(10-16 for the ALog BottleLogger) is the number of bits of ADC resolution used (oversampling for >10)

Example:

```
alog.HM1500LF_humidity_with_external_temperature1,10000,3950,10000,25,1,12);
```

2.1.2.16 Honeywell_HSC_analog()

```
float ALog::Honeywell_HSC_analog (
             int pin,
             float Vsupply,
             float Vref,
             float Pmin,
             float Pmax,
             int TransferFunction_number,
             int units,
             uint8_t ADC_resolution_nbits = 14 )
```

Cost-effective pressure sensor from Honeywell

Datasheet: http://sensing.honeywell.com/index.php?ci_id=151133

See also the **Honeywell_HSC_analog** example.

Parameters

pin	Analog pin number
Vsupply	Supply voltage to sensor
Vref	is the reference voltage of the ADC; on the ALog, this is a precision 3.3V regulator (unless a special unit without this regulator is ordered; the regulator uses significant power)
Pmin	Minimum pressure in range of sensor
Pmax	Maximum pressure in range of sensor
Pmax	Maximum pressure in range of sensor
TransferFunction_number	1, 2, 3, or 4: which transfer function is used to convert voltage to pressure • TransferFunction: 1 = 10% to 90% of Vsupply ("A" in second to last digit of part number) • TransferFunction: 2 = 5% to 95% of Vsupply ("A" in second to last digit of part number) • TransferFunction: 3 = 5% to 85% of Vsupply ("A" in second to last digit of part number) • TransferFunction: 4 = 4% to 94% of Vsupply ("A" in second to last digit of part number)
units	Output units • Units: 0 = mbar • Units: 1 = bar • Units: 2 = Pa • Units: 3 = KPa • Units: 4 = MPa • Units: 5 = inH2O • Units: 6 = PSI
ADC_resolution_nbits	(10-16 for the ALog BottleLogger) is the number of bits of ADC resolution used (oversampling for >10 bits)

Example:

```
alog.Honeywell_HSC_analog(A1, 5, 3.3, 0, 30, 1, 6);
```

2.1.2.17 HTM2500LF_humidity_temperature()

```
void ALog::HTM2500LF_humidity_temperature (
            uint8_t humidPin,
```

```
            uint8_t thermPin,
            float Rref_therm,
            uint8_t ADC_resolution_nbits = 14 )
```

HTM2500LF Relative humidity and temperature sensor

This function measures the relative humidity of using a HTM2500 tempurature and relative humidity module. The relative humidity and temperature is measured using a 14 bit oversampling method. Results are displayed on the serial monitor and saved onto the SD card to four decimal places.

Parameters

humidPin	is the analog pin connected to the humidity output voltage of the module.
thermPin	is the analog pin connected to the tempurature output voltage of the module.
Rref_therm	is the value of the reference resistor that you use with the built-in thermistor (reference resistor supplied separately, placed in appropriate slot in header)
ADC_resolution_nbits	(10-16 for the ALog BottleLogger) is the number of bits of ADC resolution used (oversampling for >10)

Example:

```
alog.HTM2500LF_humidity_temperature(1, 2, ?);
```

This function is designed for ratiometric operation – that is, the humidity sensor must be powered by the same voltage regulator that is connected to the the analog reference pin – for the ALog v2.0, this is a high-precision 3V3 regulator.

2.1.2.18 Inclinometer_SCA100T_D02_analog_Tcorr()

```
void ALog::Inclinometer_SCA100T_D02_analog_Tcorr (
            uint8_t xPin,
            uint8_t yPin,
            float Vref,
            float Vsupply,
            float R0_therm,
            float B_therm,
            float Rref_therm,
            float T0degC_therm,
            uint8_t thermPin_therm,
            uint8_t ADC_resolution_nbits = 14 )
```

Inclinometer, including temperature correction from an external sensor.

- +/- 90 degree inclinometer, measures +/- 1.0g
- Needs 4.75–5.25V input (Vsupply)
- In typical usage, turned on and off by a switching 5V charge pump or boost converter

Parameters

xPin	Analog pin number corresponding to x-oriented tilts
yPin	Analog pin number corresponding to y-oriented tilts
Vref	is the reference voltage of the analog-digital comparator; it is 3.3V on the ALog.
Vsupply	is the input voltage that drives the sensor, and is typically between 3.3 and 5V.
R0_therm	is a thermistor calibration.
B_therm	is the B- or - parameter of the thermistor.
Rref_therm	is the resistance of the corresponding reference resistor for that analog pin.
T0degC_therm	is a thermistor calibration.
thermPin_therm	is the analog pin connected to the tempurature output voltage of the module.
ADC_resolution_nbits	(10-16 for the ALog BottleLogger) is the number of bits of ADC resolution used (oversampling for >10). It is applied to both the inclinomter and its temperature correction

Example:

```
alog.Inclinometer_SCA100T_D02_analog_Tcorr(6, 2, 3.285, 5.191, \
     10080.4120953, 3298.34232031, 10000, 25, 0);
```

2.1.2.19 initialize()

```
void ALog::initialize (
             char * _logger_name,
             char * _datafilename,
             int _hourInterval,
             int _minInterval,
             int _secInterval,
             bool _ext_int = false,
             bool _LOG_ALL_SENSORS_ON_BUCKET_TIP = false )
```

Pass all variables needed to initialize logging.

Parameters

_logger_name	Name associated with this data logger; often helps to relate it to the project or site
_datafilename	Name of main data file saved to SD card; often helps to relate it to the project or site; used to be limited to 8.3 file naming convention, but now strictly is not. (I still use 8.3 names for safety's sake!)
_hourInterval	How many hours to wait before logging again; can range from 0-24.
_minInterval	How many minutes to wait before logging again; can range from 0-59.
_secInterval	How many seconds to wait before logging again; can range from 0-59.

If all time-setting functions are 0, then the logger will not sleep, and instead will log continuously. This sets the flag "_use_sleep_mode" to be false.

Parameters

_ext_int	External interrupt, set to be a tipping-bucket rain gauge, that triggers event-based logging of a timestamp
_LOG_ALL_SENSORS_ON_BUCKET_TIP	Flag that tells the logger to read every sensor when the bucket tips (if _ext_int is true) and write their outputs to "datafile" (i.e. the main data file whose name you specify with **_filename**; this is in addition to writing the timestamp of the rain gauge bucket tip.

ALog Data logger model does not need to be set: it is automatically determined from the MCU type and is used to modify pinout-dependent functions.

Example:

```
\\ Log every five minutes
alog.initialize('TestLogger01', 'lab_bench_test.alog', 0, 0, 5, 0);
```

2.1.2.20 linearPotentiometer()

```
void ALog::linearPotentiometer (
            uint8_t linpotPin,
            float Rref,
            float slope,
            char * _distance_units,
            float intercept = 0,
            uint8_t ADC_resolution_nbits = 14,
            bool Rref_on_GND_side = true )
```

Linear potentiometer (radio tuner) to measure distance

Distance based on resistance in a sliding potentiometer whose resistance may be described as a linear function

Parameters

linpotPin	Analog pin number
Rref	Resistance value of reference resistor [ohms]
slope	Slope of the line (distance = (slope)R + R0)
intercept	(R0) of the line (distance = (slope)R + R0)
ADC_resolution_nbits	(10-16 for the ALog BottleLogger) is the number of bits of ADC resolution used (oversampling for >10 bits)
_distance_units	is the name of the units of distance that are used in the linear calibration equation, and are therefore the units of this function's output.
Rref_on_GND_side	indicates the configuration of the voltage divider. True if using Alog provided Reference resistor terminals. If false, the reference resitor must be instead connected via the screw terminals. This is set true for external sensors that are built to require a VCC-side reference resistor.

The output units will be whatever you have used to create your linear calibration equation

Example:

```
// Using a 0-10k ohm radio tuner with units in mm and a perfect intercept;
// maintaining default 14-bit readings with standard-side (ALog header)
// reference resistor set-up
alog.linearPotentiometer(A0, 5000, 0.0008);
```

2.1.2.21 maxbotixHRXL_WR_analog()

```
void ALog::maxbotixHRXL_WR_analog (
             uint8_t nping = 10,
             uint8_t sonicPin = A0,
             uint8_t EX = 99,
             bool writeAll = true,
             uint8_t ADC_resolution_nbits = 10 )
```

Newer 1-mm precision MaxBotix rangefinders: analog readings

This function measures the distance between the ultrasonic sensor and an \ acoustically-reflective surface, typically water or snow. Measures distance in milimeters. Results are displayed on the serial monitor and saved onto the SD card.

Parameters

nping	is the number of range readings to take (number of pings). The mean range will be calculated and output to the serial monitor and SD card followed by the standard deviation.
sonicPin	is the analog input channel hooked up to the maxbotix sensor.
EX	is a digital output pin used for an excitation pulse. If maxbotix sensor is continuously powered, a reading will be taken when this pin is flashed high. Set to '99' if excitation pulse is not needed.
writeAll	will write each reading of the sensor (each ping) to the serial monitor and SD card.
ADC_resolution_nbits	(10-16 for the ALog BottleLogger) is the number of bits of ADC resolution used (oversampling for $>$10 bits)

Example:

```
alog.maxbotixHRXL_WR_analog(10,A2,99,0);
```

Note that sensor should be mounted away from supporting structure. These are the standard recommendations:

- For a mast that is 5 meters high (or higher) the sensor should be mounted at least 100cm away from the mast.
- For a mast that is 2.5 meters high (or lower) the sensor should be at least 75cm away from the mast.

However, in our tests, the sensors with filtering algorithms function perfectly well even when positioned close to the mast, and a short mast increases the rigidity of the installation. This was tested in the lab by placing the MaxBotix sensor flush with table legs and testing distance readings to the floor.

2.1.2.22 maxbotixHRXL_WR_Serial()

```
float ALog::maxbotixHRXL_WR_Serial (
            uint8_t Ex,
            uint8_t npings,
            bool writeAll,
            int maxRange,
            bool RS232 = false )
```

Uses the UART interface to record data from a MaxBotix sensor.

NOTE: THIS HAS CUASED LOGGERS TO FREEZE IN THE PAST; WHILE IT IS QUITE LIKELY THAT THE ISSUE IS NOW SOLVED, MORE TESTING IS REQUIRED. (ADW, 26 NOVEMBER 2016) (maybe solved w/ HW Serial?)

Parameters

Ex	Excitation pin that turns the sensor on; if this is not needed (i.e. you are turning main power off and on instead), then just set this to a value that is not a pin, and ensure that you turn the power to the sensor off and on outside of this function
npings	Number of pings over which you average; each ping itself includes ten short readings that the sensor internally processes
writeAll	will write each reading of the sensor (each ping) to the serial monitor and SD card.
maxRange	The range (in mm) at which the logger maxes out; this will be remembered to check for errors and to become a nodata values
RS232	this is set true if you use inverse (i.e. RS232-style) logic; it works at standard logger voltages (i.e. it is not true RS232). If false, TTL logic will be used.

Example:

```
// Digital pin 7 controlling sensor excitation, averaging over 10 pings,
// not recording the results of each ping, and with a maximum range of
// 5000 mm using standard TTL logic
alog.maxbotixHRXL_WR_Serial(7, 10, false, 5000, false);
```

2.1.2.23 Pyranometer()

```
void ALog::Pyranometer (
            uint8_t analogPin,
            float raw_mV_per_W_per_m2,
            float gain,
            float V_ref,
            uint8_t ADC_resolution_nbits = 14 )
```

Pyranometer wtih instrumentation amplifier

Pyranomiter is from Kipp and Zonen

nominal raw_output_per_W_per_m2_in_mV = 10./1000.; // 10 mV at 1000 W/m∗∗2

Actual raw output is based on calibration.

Parameters

analogPin	is the pin that receives the amplified voltage input
raw_mV_per_W_per_m2	is the conversion factor of the pyranometer: number of millivolts per (watt/meter^2). This does not include amplification!
gain	is the amplification factor
Vref	is the reference voltage of the ADC; on the ALog, this is a precision 3.3V regulator (unless a special unit without this regulator is ordered; the regulator uses significant power)
ADC_resolution_nbits	(10-16 for the ALog BottleLogger) is the number of bits of ADC resolution used (oversampling for >10 bits)

Example:

```
// Using precision voltage reference and 16-bit resolution (highest
// defensible oversampling resolution)
alog.Pyranometer(A0, 0.0136, 120, 3.300, 16);
```

2.1.2.24 readPin()

```
float ALog::readPin (
            uint8_t pin )
```

Read the analog value of a pin.

This function returns the analog to digital converter value (0 - 1023). Results are displayed on the serial monitor and saved onto the SD card.

Parameters

pin	is the analog pin number to be read.

Example:

```
alog.readPin(2);
```

2.1.2.25 readPinOversample()

```
float ALog::readPinOversample (
            uint8_t pin,
            uint8_t adc_bits )
```

Read the analog value of a pin, with extra resolution from oversampling

This function incorporates oversampling to extend the ADC precision past ten bits by taking more readings and statistically combing them. Results are displayed on the serial monitor and saved onto the SD card.

Parameters

pin	is the analog pin number to be read.
adc_bits	is the reading precision in bits (2^adc_bits). The ATMega328 (Arduino Uno and ALog BottleLogger core chip) has a base ADC precision of 10 bits (returns values of 0-1023) A reasonable maximum precision gain is (base_value_bits)+6, so 16 bits is a reasonable maximum precision for the ALog BottleLogger.

Example:

```
alog.readPinOversample(2, 12);
```

Output values will range from 0-1023, but be floating-point.

Readings that require more bits of precision will take longer.

2.1.2.26 sensorPowerOff()

```
void ALog::sensorPowerOff ( )
```

Turn OFF power to 3V3 regulator that connects to screw terminals.

This cuts 3V3 power to external devices.

2.1.2.27 sensorPowerOn()

```
void ALog::sensorPowerOn ( )
```

Turn ON power to 3V3 regulator that connects to screw terminals.

This powers external devices.

2.1.2.28 set_LEDpin()

```
void ALog::set_LEDpin (
            int8_t _pin )
```

Set which pin to use for the main indicator LED.

Run this, if needed, before setupLogger()

2.1.2.29 set_RTCpowerPin()

```
void ALog::set_RTCpowerPin (
            int8_t _pin )
```

Set which pin activates the 3V3 regulator to power the RTC (real-time clock).

- Set to -1 if not being used
- Set to the same as SDpowerPin if these are connected (standard for the ALog BottleLogger) Run this, if needed, before setupLogger()

2.1.2.30 set_SDpowerPin()

```
void ALog::set_SDpowerPin (
            int8_t _pin )
```

Set which pin activates the 3V3 regulator to power the SD card.

- Set to -1 if not being used
- Set to the same as RTCpowerPin if these are connected (standard for the ALog BottleLogger) Run this, if needed, before setupLogger()

2.1.2.31 set_SensorPowerPin()

```
void ALog::set_SensorPowerPin (
            int8_t _pin )
```

Set which pin activates the 3V3 regulator to power sensors and any other external 3V3 devices that receive power from the ALog's 3V3 regulator.

- Set to -1 if not being used
- Otherwise, set to the number of the pin controlling the 3V3 regulator that goes to the sensors and other peripherals. Run this, if needed, before setupLogger()

2.1.2.32 setupLogger()

```
void ALog::setupLogger ( )
```

Readies the ALog to begin measurements

Sets all pins, alarms, clock, SD card, etc: everything needed for the ALog to run properly.

2.1.2.33 sleep()

```
void ALog::sleep ( )
```

Puts the ALog data logger into a low-power sleep mode

Sets the "IS_LOGGING" flag to false, disables the watchdog timer, and puts the logger to sleep.

2.1.2.34 startAnalog()

```
void ALog::startAnalog ( )
```

Turn on power to analog sensors

Actually just turns on 3V3 regulator that could power anything. Keeping this around for backwards compatibility; look to sensorPowerOn() for the preferred version of this.

2.1.2.35 startLogging()

```
void ALog::startLogging ( )
```

Wakes the logger and starts logging

Wakes the logger: sets the watchdog timer (a failsafe in case the logger hangs), checks and clears alarm flags, looks for rain gauge bucket tips (if they occur during the middle of a logging event (ignore) or if they include a command to read all sensors with a tip), and starts to log to "datafile", if it can.

If the logger cannot reach the SD card, it sends out an LED warning message of 20 rapid flashes.

2.1.2.36 thermistorB()

```
float ALog::thermistorB (
            float R0,
            float B,
            float Rref,
            float T0degC,
            uint8_t thermPin,
            uint8_t ADC_resolution_nbits = 14,
            bool Rref_on_GND_side = true,
            bool oversample_debug = false,
            bool record_results = true )
```

Read the analog value of a pin, with extra resolution from oversampling

This function measures temperature using a thermistor characterised with the B (or) parameter equation, which is a simplification of the Steinhart-Hart equation

The function compares the thermistor risistance with the reference resistor using a voltage divider.

It returns a float of the temperature in degrees celsius. Results are displayed on the serial monitor and saved onto the SD card to four decimal places.

Parameters

R0	is the resistance of the thermistor at the known temperature
T0degC.	[]
B	is the parameter of the thermistor. [K]
Rref	is the resistance of the corresponding reference resistor for \ the analog pin set by **ThermPin** (below). []
T0degC	is the temperature at which **R0** was calibrated. [℃]
thermPin	is the analog pin number to be read. [-]
ADC_resolution_nbits	(10-16 for the ALog BottleLogger) is the number of bits of ADC resolution used (oversampling for >10 bits) [bits]
Rref_on_GND_side	indicates the configuration of the voltage divider. True if using Alog provided Reference resistor terminals. If false, the reference resitor must be instead connected via the screw terminals. This is set true for external sensors that are built to require a VCC-side reference resistor.
oversample_debug	is true if you want a separate file, "Oversample.txt", to record every individual reading used in the oversampling.
record_results	is true if you want to save results to the SD card and print to the serial monitor.

Examples:

```
// Cantherm from Digikey
// 10 kOhm @ 25degC, 3950 K b-value, 30 kOhm reference resistor, on
// analog Pin 1, 14-bit precision
alog.thermistorB(10000, 3950, 30000, 25, 2, 14);
// EPCOS, DigiKey # 495-2153-ND
// 10 kOhm @ 25degC, 3988 K b-value, 13.32 kOhm reference resistor, on
// analog Pin 1, 12-bit precision
alog.thermistorB(10000, 3988, 13320, 25, 1, 12);
```

2.1.2.37 ultrasonicMB_analog_1cm()

```
void ALog::ultrasonicMB_analog_1cm (
              uint8_t nping,
              uint8_t EX,
              uint8_t sonicPin,
              bool writeAll )
```

Old 1-cm resolution Maxbotix ultrasonic rangefinders: analog measurements

This function measures the distance between the ultrasonic sensor and an acustically reflective surface, typically water or snow. Measures distance in centimeters. Results are displayed on the serial monitor and saved onto the SD card.

This is for the older MaxBotix sensors, whose maximum precision is in centimeters.

Parameters

nping	is the number of range readings to take (number of pings). The mean range will be calculated and output to the serial monitor and SD card followed by the standard deviation.
EX	is a digital output pin used for an excitation pulse. If maxbotix sensor is continuously powered a reading will be taken when this pin is flashed high. Set to '99' if excitation pulse is not needed.
sonicPin	is the analog input channel hooked up to the maxbotix sensor.
writeAll	will write each reading of the sensor (each ping) to the serial monitor and SD card.

Example:

```
alog.ultrasonicMB_analog_1cm(10, 99, 2, 0);
```

Note that sensor should be mounted away from supporting structure. For a mast that is 5 meters high (or higher) the sensor should be mounted at least 100cm away from the mast. For a mast that is 2.5 meters high (or lower) the sensor should be at least 75cm away from the mast.

2.1.2.38 vdivR()

```
void ALog::vdivR (
            uint8_t pin,
            float Rref,
            uint8_t ADC_resolution_nbits = 10,
            bool Rref_on_GND_side = true )
```

Resistance from a simple voltage divider

Parameters

pin	Analog pin number
Rref	Resistance value of reference resistor [ohms]
ADC_resolution_nbits	(10-16 for the ALog BottleLogger) is the number of bits of ADC resolution used (oversampling for $>$10 bits)
Rref_on_GND_side	indicates the configuration of the voltage divider. True if using Alog provided Reference resistor terminals. If false, the reference resitor must be instead connected via the screw terminals. This is set true for external sensors that are built to require a VCC-side reference resistor.

Example:

```
// Use standard reference resistor headers: let last parameter be false
// (default)
alog.vdivR(A2, 10000, 12);
```

2.1.2.39 Wind_Vane_Inspeed()

```
void ALog::Wind_Vane_Inspeed (
            uint8_t vanePin )
```

Wind vane: resistance changes with angle to wind.

Parameters

vanePin	is the analog pin that reads the wind vane resistance

This function is specialized for the Inspeed eVane2. Here, a resistance of 0 equates to wind from the north, and resistence increases in a clockwise direction.

Connect one wire to power supply, one wire to analog pin, one wire to GND

From documentation:

- 5 - 95% of power supply input voltage = 0 to 360 degrees of rotation.
- Uses Hall Effect Sensor
- Don't forget to use set screw to zero wind sensor before starting!

Example:

```
// After setting up and zeroing the eVane to North, you wire it to
// analog pin 7 on the ALog
alog.Wind_Vane_Inspeed(A7);
```

The documentation for this class was generated from the following files:

- src/ALog.h
- src/ALog.cpp

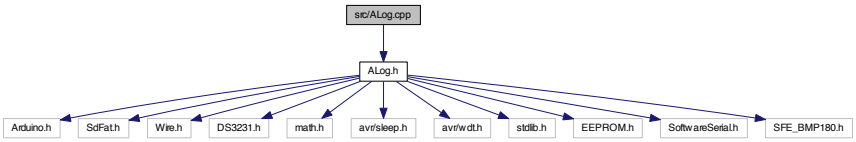
Chapter 3

File Documentation

3.1 src/ALog.cpp File Reference

```
#include <ALog.h>
```

Include dependency graph for ALog.cpp:



Macros

- #define **cbi**(sfr, bit) (_SFR_BYTE(sfr) &= ~_BV(bit))
- #define **sbi**(sfr, bit) (_SFR_BYTE(sfr) |= _BV(bit))

Functions

- void **wakeUpNow** ()
- void **wakeUpNow_tip** ()
- void **_ISR_void** ()
- void **_anemometer_count_increment** ()
- void save_Aref (float _V)
- float read_Aref ()
- void **_internalDateTime** (uint16_t *date, uint16_t *time)

Variables

- const bool **h12** = false
- uint8_t **hourInterval**
- uint8_t **minInterval**
- uint8_t **secInterval**
- uint8_t **_hours**
- uint8_t **_minutes**
- uint8_t **_seconds**
- bool **_use_sleep_mode** = true
- bool **CAMERA_IS_ON** = false
- bool **IS_LOGGING** = false
- char * **datafilename**
- char * **logger_name**
- bool **extInt**
- bool **NEW_RAIN_BUCKET_TIP** = false
- bool **LOG_ALL_SENSORS_ON_BUCKET_TIP**
- bool **first_log_after_booting_up** = true
- uint8_t **rotation_count** = 0
- RTClib **RTC**
- DS3231 **Clock**
- SdFat **sd**
- SdFile **datafile**
- SdFile **otherfile**
- SdFile **headerfile**
- DateTime **now**

3.1.1 Detailed Description

Data logger library Designed for the ALog Modules should work for any Arduino-based board with minimal modificiation Goals: (1) Manage logger utility functions, largely behind-the-scenes (2) Simplify data logger operations to one-line calls

Written by Andy Wickert, 2011-2017, and Chad Sandell, 2016-2017 Started 27 September 2011

Designed to greatly simplify Arduino sketches for the ALog and reduce what the end user needs to do into relatively simple one-line calls.

LICENSE: GNU GPL v3

ALog.cpp is part of ALog, an Arduino library written by Andrew D. Wickert and Chad T. Sandell Copyright (C) 2011-2017, Andrew D. Wickert Copyright (C) 2016-2017, Chad T. Sandell Copyright (C) 2016-2017, Regents of the University of Minnesota

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see http://www.gnu.org/licenses/.

3.1.2 Function Documentation

3.1.2.1 read_Aref()

```
float read_Aref ( )
```

Read the analog (ADC) sensor reference voltage from EEPROM, return float.

ADC == analog-digital comparator EEPROM = permanent memory (persists after shutdown) See: https://www.↩ arduino.cc/en/Reference/EEPROMGet

Example:

```
float Vref;
Vref = alog.readAref();
```

3.1.2.2 save_Aref()

```
void save_Aref (
            float _V )
```

Saves a float as the reference voltage for the ADC ("Vref") to the EEPROM

ADC == analog-digital comparator EEPROM = permanent memory (persists after shutdown) See: https://www.↩ arduino.cc/en/Reference/EEPROMPut

This function is only called rarely, as this value is typically measured only once.

Parameters

↩ _↩ *V*	reference voltage, ideally measured under load [V]

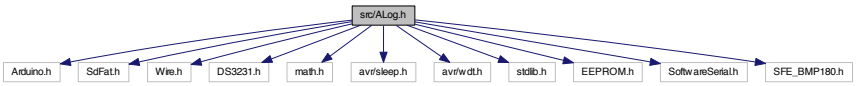
Example:

```
// Measuring 3.297V with a calibrated multimeter between 3V3 and GND
// Then:
alog.saveAref(3.297);
```

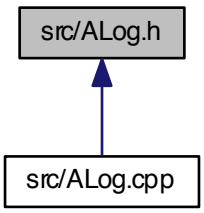
3.2 src/ALog.h File Reference

```
#include <Arduino.h>
#include <SdFat.h>
#include <Wire.h>
#include <DS3231.h>
#include <math.h>
#include <avr/sleep.h>
#include <avr/wdt.h>
#include <stdlib.h>
#include <EEPROM.h>
#include <SoftwareSerial.h>
#include <SFE_BMP180.h>
```

Include dependency graph for ALog.h:



This graph shows which files directly or indirectly include this file:



Classes

- class ALog

Functions

- void **wakeUpNow** ()
- void **wakeUpNow_tip** ()
- void **_ISR_void** ()
- void **_anemometer_count_increment** ()
- void **_internalDateTime** (uint16_t *date, uint16_t *time)

3.2.1 Detailed Description

ALog.h

Data logger library header
Designed for the ALog
Modules should work for any Arduino-based board with minimal modificiation.
Goals:

1. Manage logger utility functions, largely behind-the-scenes
2. Simplify data logger operations to one-line calls

Written by Andy Wickert, 2011-2017, and Chad Sandell, 2017
Started 27 September 2011

Designed to greatly simplify Arduino sketches for the ALog and reduce what the end user needs to do into relatively simple one-line calls.

LICENSE: GNU GPL v3

Index

_sensor_function_template
    ALog, 57

ALog, 55
    _sensor_function_template, 57
    ALog, 56
    analogReadOversample, 57
    Anemometer_reed_switch, 58
    Barometer_BMP180, 59
    Decagon5TE, 59
    DecagonGS1, 60
    endAnalog, 60
    endLogging, 60
    get_3V3_measured_voltage, 60
    get_5V_measured_voltage, 61
    get_serial_number, 61
    get_use_sleep_mode, 61
    goToSleep_if_needed, 61
    HM1500LF_humidity_with_external_temperature, 62
    HTM2500LF_humidity_temperature, 64
    HackHD, 61
    Honeywell_HSC_analog, 63
    Inclinometer_SCA100T_D02_analog_Tcorr, 65
    initialize, 66
    linearPotentiometer, 67
    maxbotixHRXL_WR_Serial, 68
    maxbotixHRXL_WR_analog, 68
    Pyranometer, 69
    readPin, 70
    readPinOversample, 70
    sensorPowerOff, 72
    sensorPowerOn, 72
    set_LEDpin, 72
    set_RTCpowerPin, 72
    set_SDpowerPin, 73
    set_SensorPowerPin, 73
    setupLogger, 73
    sleep, 74
    startAnalog, 74
    startLogging, 74
    thermistorB, 74
    ultrasonicMB_analog_1cm, 75
    vdivR, 76
    Wind_Vane_Inspeed, 76
ALog.cpp
    read_Aref, 81
    save_Aref, 81
analogReadOversample
    ALog, 57
Anemometer_reed_switch
    ALog, 58

Barometer_BMP180
    ALog, 59

Decagon5TE
    ALog, 59
DecagonGS1
    ALog, 60

endAnalog
    ALog, 60
endLogging
    ALog, 60

get_3V3_measured_voltage
    ALog, 60
get_5V_measured_voltage
    ALog, 61
get_serial_number
    ALog, 61
get_use_sleep_mode
    ALog, 61
goToSleep_if_needed
    ALog, 61

HM1500LF_humidity_with_external_temperature
    ALog, 62
HTM2500LF_humidity_temperature
    ALog, 64
HackHD
    ALog, 61
Honeywell_HSC_analog
    ALog, 63

Inclinometer_SCA100T_D02_analog_Tcorr
    ALog, 65
initialize
    ALog, 66

linearPotentiometer
    ALog, 67

maxbotixHRXL_WR_Serial
ALog, 68
maxbotixHRXL_WR_analog
ALog, 68

Pyranometer
ALog, 69

read_Aref
ALog.cpp, 81
readPin
ALog, 70
readPinOversample
ALog, 70

save_Aref
ALog.cpp, 81
sensorPowerOff
ALog, 72
sensorPowerOn
ALog, 72
set_LEDpin
ALog, 72
set_RTCpowerPin
ALog, 72
set_SDpowerPin
ALog, 73
set_SensorPowerPin
ALog, 73
setupLogger
ALog, 73
sleep
ALog, 74
src/ALog.cpp, 79
src/ALog.h, 82
startAnalog
ALog, 74
startLogging
ALog, 74

thermistorB
ALog, 74

ultrasonicMB_analog_1cm
ALog, 75

vdivR
ALog, 76

Wind_Vane_Inspeed
ALog, 76