Logger (for ALog) v0.2 for ALog 2.0

Generated by Doxygen 1.8.11

# **Contents**

1	ALo	g Guide		1
	1.1	Introdu	action	1
	1.2	Softwa	re Installation	1
		1.2.1	ALog program and libraries	1
		1.2.2	Arduino	2
	1.3	Quick-	start guide	2
		1.3.1	Installing requisite software	2
		1.3.2	Upload the program "alog_no_sensors"	2
		1.3.3	Interfacing with the ALog and setting its clock	6
		1.3.4	Creating and uploading a custom data logging routine	6
		1.3.5	Attaching sensors	6
		1.3.6	Writing your own sensor code	6
		1.3.7	Field deployment	6
	1.4	ALog p	pinout and peripherals	7
	1.5	About		7
2	Logg	ner		9
_	Logi	yei		3
3	Clas	s Index		11
	3.1	Class I	ist	11
4	File	Index		13
	<b>4</b> 1	File I is	et	12

iv CONTENTS

5	Clas	s Docu	mentation		15
	5.1	Logge	r Class Ref	ference	15
		5.1.1	Construc	tor & Destructor Documentation	16
			5.1.1.1	Logger()	16
		5.1.2	Member	Function Documentation	16
			5.1.2.1	analogReadOversample(int pin, int adc_bits=10, int nsamples=1, debug=false) .	16
			5.1.2.2	Anemometer_reed_switch(int interrupt_pin_number, unsigned long reading_← duration_milliseconds, float meters_per_second_per_rotation)	17
			5.1.2.3	AtlasScientific(char *command, int softSerRX=6, int softSerTX=7, uint32_← t baudRate=38400, bool printReturn=true, bool saveReturn=true)	17
			5.1.2.4	Decagon5TE(int excitPin, int dataPin)	18
			5.1.2.5	DecagonGS1(int pin, float Vref, uint8_t ADC_resolution_nbits=14)	18
			5.1.2.6	endAnalog()	19
			5.1.2.7	endLogging()	19
			5.1.2.8	get_use_sleep_mode()	19
			5.1.2.9	HackHD(int control_pin, bool want_camera_on)	19
			5.1.2.10	HM1500LF_humidity_with_external_temperature(int humidPin, float Vref, float R0, float B, float Rref, float T0degC, int thermPin, uint8_t ADC_resolution_nbits=14)	) 20
			5.1.2.11	Honeywell_HSC_analog(float Vsupply, float Pmin, float Pmax, int Transfer ← Function_number, int units, int pin, uint8_t ADC_resolution_nbits=14)	21
			5.1.2.12	HTM2500LF_humidity_temperature(int humidPin, int thermPin, float Rref_therm, uint8_t ADC_resolution_nbits=14)	22
			5.1.2.13	Inclinometer_SCA100T_D02_analog_Tcorr(int xPin, int yPin, float Vref, float Vsupply, float R0, float B, float Rref, float T0degC, int thermPin, uint8_t ADC← _resolution_nbits=14)	22
			5.1.2.14	initialize(char *_logger_name, char *_sitecode, int _dayInterval, int _hourInterval, int _minInterval, int _secInterval, bool _ext_int=false, bool _LOG_ON_BUCKE ← T_TIP=false)	23
			5.1.2.15	linearPotentiometer(int linpotPin, float Rref, float slope, float intercept=0, uint8_t ADC_resolution_nbits=14)	23
			5.1.2.16	maxbotixHRXL_WR_analog(int nping=10, int sonicPin=A0, int EX=99, bool writeAll=true, uint8_t ADC_resolution_nbits=13)	24
			5.1.2.17	maxbotixHRXL_WR_Serial(int Ex, int nping, bool writeAll, int maxRange, bool RS232=false)	25
			5.1.2.18	Pyranometer(int analogPin, float raw_mV_per_W_per_m2, float gain, float V_ref, uint8_t ADC_resolution_nbits=14)	25

CONTENTS

		5.1.2.19	readPin(int pin)	26
		5.1.2.20	readPinOversample(int pin, int bits)	26
		5.1.2.21	setupLogger()	27
		5.1.2.22	sleep()	27
		5.1.2.23	startAnalog()	27
		5.1.2.24	startLogging()	27
		5.1.2.25	thermistorB(float R0, float B, float Rref, float T0degC, int thermPin, uint8 $_\leftarrow$ t ADC_resolution_nbits=14, bool Rref_on_GND_side=true;bool oversample $_\leftarrow$ debug=false)	27
		5.1.2.26	ultrasonicMB_analog_1cm(int nping, int EX, int sonicPin, bool writeAll)	28
		5.1.2.27	vdivR(int pin, float Rref, uint8_t ADC_resolution_nbits=10, bool Rref_on_GND⊷ _side=true)	28
		5.1.2.28	Wind_Vane_Inspeed(int vanePin)	29
6	File	Documentation		31
	6.1	Logger.cpp File F	Reference	31
		6.1.1 Detailed	Description	32
	6.2	Logger.h File Ref	erence	33
		6.2.1 Detailed	Description	34

# **Chapter 1**

# **ALog Guide**

#### 1.1 Introduction

The **ALog** data logger is an Arduino-based data logger that includes a full software library and documentation. The goal of this documentation is to ease the transition of researchers from closed-source proprietary technologies that are often well-documented, but can be expensive black boxes, to a fully open-source hardware and software toolchain that gives researchers all the power to control their data collection. For citizen-scientists who are interested in this tool: welcome, and it is greatly hoped that these instructions will suffice to get you started on the road to making measurements of our natural environment.

### 1.2 Software Installation

All of our software is available from our GitHub repository, https://github.com/NorthernWidget. To install the Arduino software and the core libraries, follow these steps:

## 1.2.1 ALog program and libraries

- Download Arduino's latest IDE (version 1.6.9 at the time of writing): http://arduino.cc/en/Main/

  Software
- 2. Navgate to your **sketchbook** folder in your home directory, and into the **libraries** folder that it contains. Clone or download the following git repositories there:
- 1. 1. The main Logger library: https://github.com/NorthernWidget/Logger
- 1. 2. The DS3231 clock library: https://github.com/NorthernWidget/DS3231.
- 1. 3. SdFat library: https://github.com/greiman/SdFat
- 2. In order to communicate with the ALog outside of the Arduino IDE (step 1) and set its clock, you will want to download ALogTalk from our GitHub repository, https://github.com/NorthernWidget/ALog← Talk, along with Python if you are running Windows (Linux and Mac users have Python by default). See http://www.python.org/getit/.

Once all of these are installed, you are ready to program your ALog. Our Quick Start Guide is in the works, and will provide the next steps. Please e-mail info@northernwidget.com (for the full team) or Chad Sandell (sand0724@umn.edu) if you have questions

2 ALog Guide

### 1.2.2 Arduino

Limor Fried has an excellent tutorial for Arduino at http://www.ladyada.net/learn/arduino/. If you are unfamiliar with the platform (and/or embedded electronics), I strongly suggest that you purchase and Arduino Uno (http://arduino.cc/en/Main/arduinoBoardUno), which is their basic board upon which the A $\leftarrow$  Log BottleLogger is based, and run through these tutorials to get used to C/C++ programming and Arduino.

# 1.3 Quick-start guide

This section, our quick-start guide is meant to get you up and running with your ALog BottleLogger as efficiently as possible.

All right. You have your ALog data logger, and you're ready to measure... something. Anything, really. But you need a hand getting out of the blocks. This section is here for you.

## 1.3.1 Installing requisite software

Go to ALog program and libraries, above, and follow the installation directions.

# 1.3.2 Upload the program "alog\_no\_sensors"

Now, upload some code to your data logger. We'll start with an example that does nothing but log the time. Follow these steps:

- 1. Start the Arduino application on your computer.
- 2. Go to File  $\rightarrow$  Examples  $\rightarrow$  Logger  $\rightarrow$  alog\_no\_sensors. Click to open it.

1.3 Quick-start guide 3

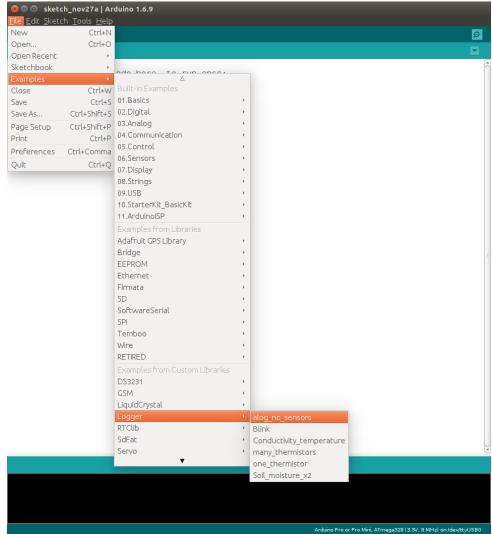


Figure 1.1 Open alog\_no\_sensors: this is our blank template upon which you can write your logger code.

For now, we will just upload this file alone.

3. Select "Arduino Pro" under "boards". Sparkfun's Arduino Pro 3.3V has the same microcontroller chip and clock speed as the ALog BottleLogger, so its settings are compatible.

4 ALog Guide

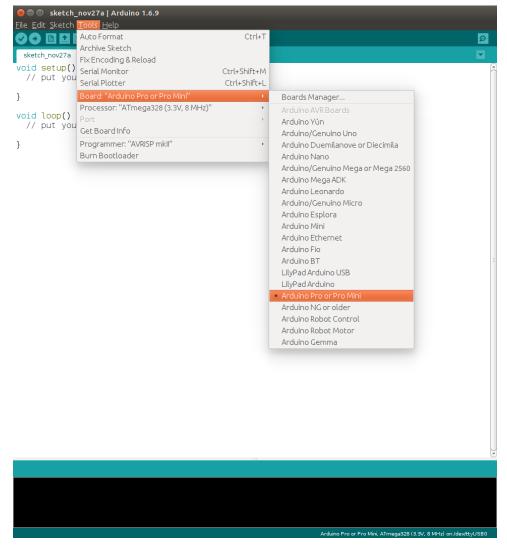


Figure 1.2 Sparkfun's Arduino pro can have the same settings as the ALog.

4. Select the processor for the ALog: 8 MHz Arduino Pro 328.

1.3 Quick-start guide 5

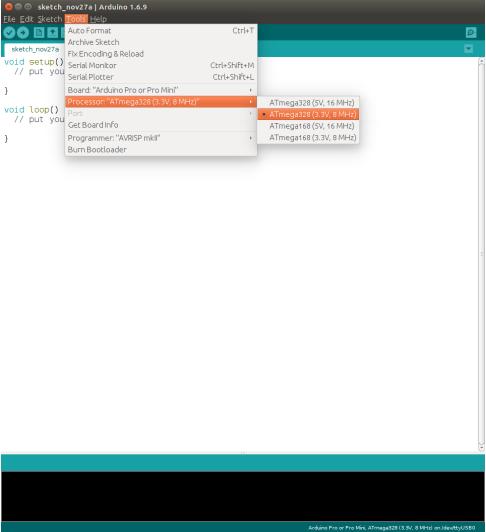


Figure 1.3 The Arduino Pro or Pro Mini (3.3V, 8 MHz) w/ ATmega328 has the proper settings for the ALog BottleLogger. Your version of Arduino won't have the 'ALog BottleLogger' custom setting at the top; it is part of our work to eventually have a more streamlined ALog programming interface.

- 5. Plug in your ALog BottleLogger using a USB A to B cable.
- 6. The BottleLogger should blink a couple of times on the LED's that are right by the USB port. This means that it sees that it has been plugged in. Sometimes, we ship the BottleLoggers with a "blink" program installed. This will cause the large red LED in the center to blink when the ALog is plugged in... so your board might do this too. It's one of our ways of testing that the board is programmable before shipping it to you.
- 7. Click on the "tools" menu, like the above figure. The "Serial Port" item in that menu should no longer be grayed out. Select your serial port.
- 8. Click "upload" from within the window holding "alog\_no\_sensors", and send the code to the ALog. It will compile for a little while first, and then be sent over as a bitstream while the two small LED's by the USB port on the ALog blink furiously: these are telling you that data is being sent to the logger (RX) or transmitted from the logger to the computer (TX).
- 9. Once this is done, click the "Serial Monitor" button to see what your data logger says. It will probably say something about setting the clock while the ALog flashes a syncopated rhythm on its main LED. This means that it is time for your next step...

6 ALog Guide

### 1.3.3 Interfacing with the ALog and setting its clock

To set the clock of the ALog, plug it into your computer and run the "ALogTalk.py" program. This requires a Python interpreter (standard on all Linux and Mac computers) and can be done from the terminal by navigating to the ALogTalk directory and doing one of two things.

First, if you have a standard

FINISH THIS AFTER WORKING THROUGH CLOCK SETTING PROGRAMS! SEE IF YOU CAN GET THE HAWNDSHAKE TO WORK!

python ALogTalk.py

Instructions will appear on the screen; if the logger does not respond, you can push its "reset" button.

python ALogTalk.py

Instructions will appear on the screen; if the logger does not respond, you can push its "reset" button.

## 1.3.4 Creating and uploading a custom data logging routine

The Github repository has examples of data logging routines, which you can modify for your own purposes. See the "examples" folder. You can also look at "Logger.h" to see what variables you need to pass to each sensor's function.

These examples include comments for where you should place your commands to communicate with various sensors, both analog and digital.

If you are unfamiliar with C or Arduino programming, the Arduino reference guide can help:  $http://arduino. \leftarrow cc/en/Reference/HomePage$ .

**IMPORTANT:** many examples of ALog code are included in this package; please use them as possible starting points for your code.

# 1.3.5 Attaching sensors

Use the screw terminals (labeled) to connect sensors to the logger. These screw terminals are labeled with pin numbers that correspond to the pin numbers used in the program.

#### 1.3.6 Writing your own sensor code

### 1.3.7 Field deployment

We have learned a bit from past field installations, so you can e-mail us at info@northernwidget.com to tell us that you want some of these examples on the website. In the meantime, it's basically an exercise in keeping the logger away from the elements and animals that like to chew wires.

# 1.4 ALog pinout and peripherals

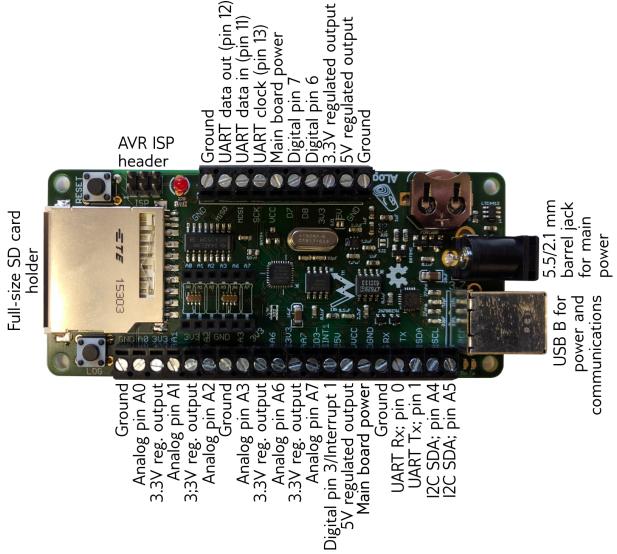


Figure 1.4 ALog pinout and peripherals.

# 1.5 About

Who is Northern Widget? What is the connection to the University of Minnesota?

Northern Widget is a company that

8 ALog Guide

# **Chapter 2**

# Logger

Data logger library for the Arduino-based ALog (http://northernwidget.com/?page\_id=8)

10 Logger

# **Chapter 3**

# **Class Index**

•					
3.	7	( -	lass		ıct
u.			เฉออ	_	เอเ

Here are the classes, structs, unions and interfaces with brief descriptions:	
Logger	15

12 Class Index

# **Chapter 4**

# File Index

# 4.1 File List

Here is a list of all documented files with brief descriptions:

Logger.cpp					 									 			 					3
Logger.h					 									 			 					3:

14 File Index

# **Chapter 5**

# **Class Documentation**

# 5.1 Logger Class Reference

#### **Public Member Functions**

- Logger ()
  - Logger library for the Arduino-based ALog data logger.
- void initialize (char \*\_logger\_name, char \*\_sitecode, int \_dayInterval, int \_hourInterval, int \_minInterval, int \_secInterval, bool \_ext\_int=false, bool \_LOG\_ON\_BUCKET\_TIP=false)
- void setupLogger ()
- void sleep ()
- · void startLogging ()
- void endLogging ()
- · void startAnalog ()
- void endAnalog ()
- bool get\_use\_sleep\_mode ()
- float readPin (int pin)
- float readPinOversample (int pin, int bits)
- float analogReadOversample (int pin, int adc\_bits=10, int nsamples=1, debug=false)
- float thermistorB (float R0, float B, float Rref, float T0degC, int thermPin, uint8\_t ADC\_resolution\_nbits=14, bool Rref\_on\_GND\_side=true;bool oversample\_debug=false)
- void ultrasonicMB\_analog\_1cm (int nping, int EX, int sonicPin, bool writeAll)
- float maxbotixHRXL WR Serial (int Ex, int nping, bool writeAll, int maxRange, bool RS232=false)
- void maxbotixHRXL\_WR\_analog (int nping=10, int sonicPin=A0, int EX=99, bool writeAll=true, uint8\_t AD
   —
   C\_resolution\_nbits=13)
- void Decagon5TE (int excitPin, int dataPin)
- void DecagonGS1 (int pin, float Vref, uint8\_t ADC\_resolution\_nbits=14)
- void vdivR (int pin, float Rref, uint8\_t ADC\_resolution\_nbits=10, bool Rref\_on\_GND\_side=true)
- void AtlasScientific (char \*command, int softSerRX=6, int softSerTX=7, uint32\_t baudRate=38400, bool printReturn=true, bool saveReturn=true)
- void HTM2500LF\_humidity\_temperature (int humidPin, int thermPin, float Rref\_therm, uint8\_t ADC\_← resolution nbits=14)
- void HM1500LF\_humidity\_with\_external\_temperature (int humidPin, float Vref, float R0, float B, float Rref, float T0degC, int thermPin, uint8 t ADC resolution nbits=14)
- void Inclinometer\_SCA100T\_D02\_analog\_Tcorr (int xPin, int yPin, float Vref, float Vsupply, float R0, float B, float Rref, float T0degC, int thermPin, uint8\_t ADC\_resolution\_nbits=14)

• void Anemometer\_reed\_switch (int interrupt\_pin\_number, unsigned long reading\_duration\_milliseconds, float meters\_per\_second\_per\_rotation)

- void Wind\_Vane\_Inspeed (int vanePin)
- void Pyranometer (int analogPin, float raw\_mV\_per\_W\_per\_m2, float gain, float V\_ref, uint8\_t ADC\_← resolution nbits=14)
- void HackHD (int control pin, bool want camera on)
- float Honeywell\_HSC\_analog (float Vsupply, float Pmin, float Pmax, int TransferFunction\_number, int units, int pin, uint8\_t ADC\_resolution\_nbits=14)

#### 5.1.1 Constructor & Destructor Documentation

```
5.1.1.1 Logger::Logger()
```

Logger library for the Arduino-based ALog data logger.

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 "Logger"

```
logger Logger();
```

#### 5.1.2 Member Function Documentation

5.1.2.1 float Logger::analogReadOversample ( int pin, int adc\_bits = 10, int nsamples = 1, 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 Analog← Read() function

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

pin	is the analog pin number
adc_bits	is the reading precision in bits (2 <sup>adc_bits</sup> ). 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.  Generated by Doxygen
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) logger.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 logger.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 = logger.analogReadOversample(A3, 14, 1);
```

5.1.2.2 void Logger::Anemometer\_reed\_switch ( int 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
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
logger.Anemometer_reed_switch(3, 4000, 1.1176);
```

5.1.2.3 void Logger::AtlasScientific ( char \* command, int softSerRX = 6, int softSerTX = 7, uint32\_t baudRate = 38400, bool printReturn = true, bool saveReturn = true)

Atlas Scientific sensors: water properties and chemistry.

Generalized serial interface for Atlas Scientific sensors. It uses SoftwareSerial AND THEREFORE IS LIKELY UN 

STABLE AFTER A FEW DAYS IN THE FIELD. The watchdog timer SHOULD BE ABLE TO CATCH THIS, but this has not yet been tested.

Code in comments is on the way to replacing this with hardware serial. It is also possible to replace this with I2C code, which will be more versitile: can connect many sensors to same I2C port, so long as they remain isolated.

#### **Parameters**

command	is the instruction code sent to the Atlas Scientific product. See the data sheet for your specific
	sensor.
softSerRx	is the software serial receive port
softSerTx	is the software serial transmit port
baudRate	is set by the Atlas Scientific sensor's baud rate.
printReturn	is true if you determines whether you care about (i.e. want to print) the Serial response, and false if you would just like to clear the buffer.
saveReturn	is true if you want to save the response to the SD card.

#### Example:

```
// read a pH probe using pins 7 and 8 as Rx and Tx, and save its results: logger.AtlasScientific("R", 7, 8);
```

5.1.2.4 void Logger::Decagon5TE ( int excitPin, int 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:

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

5.1.2.5 void Logger::DecagonGS1 (int pin, float Vref, uint8\_t ADC\_resolution\_nbits = 14)

Ruggedized Decagon Devices soil moisture sensor

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 resolutieen $\mu_{\rm by\ Doxygen}$ (oversampling for $>$ 10 bits)

#### Example:

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

5.1.2.6 void Logger::endAnalog()

Turn off power to analog sensors

5.1.2.7 void Logger::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!

```
5.1.2.8 bool Logger::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

```
5.1.2.9 void Logger::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.

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	
Generated by Doxygen	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;
\ensuremath{//} 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) {</pre>
  logger.HackHD(8, true);
  {\tt camera\_on = true; // \ Maybe \ I \ can \ get \ a \ global \ variable \ from \ this \ library}
                      \ensuremath{//} 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){
    logger.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

5.1.2.10 void Logger::HM1500LF\_humidity\_with\_external\_temperature (int humidPin, float Vref, float R0, float B, float Rref, float T0degC, int thermPin, 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 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.

humidPin	is the analog pin connected to the humidity output voltage of the module.
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)

### Example:

logger.HTM2500LF\_humidity\_temperature(1,?,10000,3950,10000,25,2);

5.1.2.11 float Logger::Honeywell\_HSC\_analog ( float *Vsupply*, float *Pmin*, float *Pmax*, int *TransferFunction\_number*, int *units*, int *pin*, uint8\_t *ADC\_resolution\_nbits* = 1 4 )

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
	<ul> <li>TransferFunction: 1 = 10% to 90% of Vsupply ("A" in second to last digit of part number)</li> </ul>
	<ul> <li>TransferFunction: 2 = 5% to 95% of Vsupply ("A" in second to last digit of part number)</li> </ul>
	<ul> <li>TransferFunction: 3 = 5% to 85% of Vsupply ("A" in second to last digit of part number)</li> </ul>
	<ul> <li>TransferFunction: 4 = 4% to 94% of Vsupply ("A" in second to last digit of part number)</li> </ul>
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:

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

5.1.2.12 void Logger::HTM2500LF\_humidity\_temperature ( int *humidPin*, int *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:

```
logger.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 analog reference pin – for the ALog v2.0, this is a high-precision 3V3 regulator.

5.1.2.13 void Logger::Inclinometer\_SCA100T\_D02\_analog\_Tcorr ( int *xPin*, int *yPin*, float *Vref*, float *Vsupply*, float *R0*, float *B*, float *Rref*, float *T0degC*, int *thermPin*, uint8\_t *ADC\_resolution\_nbits* = 1.4 )

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

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	
thermPin_therm	is the analog pin connected to the tempurature output voltage of the module.	
T0degC_therm	is a thermistor calibration. Generated by Doxyger	
Rref_therm	is the resistance of the corresponding reference resistor for that analog pin.	
B_therm	is the B- or - parameter of the thermistor.	
R0_therm	is a thermistor calibration.	
humidPin	is the analog pin connected to the humidity output voltage of the module.	
Vsupply	is the input voltage that drives the sensor, and is typically $\sim\!5$ V.	
Vref	is the reference voltage of the analog-digital comparator; it is 3.3V on the ALog.	
yPin	Analog pin number corresponding to y-oriented tilts	
xPin	Analog pin number corresponding to x-oriented tilts	

#### Example:

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

5.1.2.14 void Logger::initialize ( char \* \_logger\_name, char \* \_sitecode, int \_dayInterval, int \_hourInterval, int \_minInterval, int \_secInterval, bool \_ext\_int = false, bool \_LOG\_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
_filename	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, but now is not.
_dayInterval	How many days to wait before logging again; can range from 0-6.
_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.

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
logger.initialize('TestLogger01', 'lab_bench_test.alog', 0, 0, 5, 0);
```

5.1.2.15 void Logger::linearPotentiometer ( int *linpotPin*, float *Rref*, float *slope*, float *intercept* = 0, uint8\_t  $ADC\_resolution\_nbits = 14$  )

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)
Rref_on_GND-Side	indicates the configuration of the voltage divider. True if using Alog provided Reference resistor terminals. If false, the reference resistor 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
logger.linearPotentiometer(AO, 5000, 0.0008);
```

```
5.1.2.16 void Logger::maxbotixHRXL_WR_analog ( int nping = 10, int sonicPin = A0, int EX = 99, bool writeAll = true, uint8_t ADC_resolution_nbits = 13 )
```

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:

```
logger.ultrasonicMB_analog_1cm(10, 99, 2, 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.

5.1.2.17 float Logger::maxbotixHRXL\_WR\_Serial (int Ex, int nping, 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
logger.maxbotixHRXL_WR_Serial(7, 10, false, 5000, false);
```

5.1.2.18 void Logger::Pyranometer ( int analogPin, float  $raw_mV_per_wQ_per_m2$ , 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.

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!

### **Parameters**

gain	is the amplification factor
V_ref	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)
logger.Pyranometer(A0, 0.0136, 120, 3.300, 16);
```

### 5.1.2.19 float Logger::readPin ( int 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:

```
logger.readPin(2);
```

## 5.1.2.20 float Logger::readPinOversample ( int pin, int 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:

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

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

Readings that require more bits of precision will take longer.

```
5.1.2.21 void Logger::setupLogger ( )
```

Readies the ALog to begin measurements

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

```
5.1.2.22 void Logger::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.

```
5.1.2.23 void Logger::startAnalog ( )
```

Turn on power to analog sensors

```
5.1.2.24 void Logger::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.

```
5.1.2.25 float Logger::thermistorB (float R0, float B, float Rref, float T0degC, int thermPin, uint8_t ADC_resolution_nbits = 14, bool Rref_on_GND_side = true; bool oversample_debug=false)
```

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.

R0	is the resistance of the thermistor at the known temperature
T0degC	
В	is the parameter of the thermistor
Rref	is the resistance of the corresponding reference resistor for \ the analog pin set by
Generated by Doxygen	ThermPin (below).
T0degC	is the temperature at which <b>R0</b> 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

#### Example:

```
// Contherm from Digikey, 14-bit precision logger.thermistorB(10000, 3950, 30000, 25, 2, 14); // EPCOS, DigiKey # 495-2153-ND, 14-bit precision logger.thermistorB(10000, 3988, 13320, 25, 1, 14);
```

5.1.2.26 void Logger::ultrasonicMB\_analog\_1cm (int nping, int EX, int 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:

```
logger.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.

5.1.2.27 void Logger::vdivR ( int pin, float Rref, uint8\_t ADC\_resolution\_nbits = 10, bool Rref\_on\_GND\_side = true )

Resistance from a simple voltage divider

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)
logger.vdivR(A2, 10000, 12);

5.1.2.28 void Logger::Wind_Vane_Inspeed ( int vanePin )
```

Wind vane: resistance changes with angle to wind.

#### **Parameters**

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
logger.Wind_Vane_Inspeed(A7);
```

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

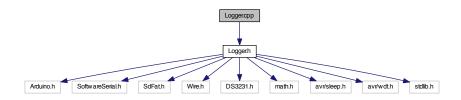
- · Logger.h
- · Logger.cpp

# **Chapter 6**

# **File Documentation**

# 6.1 Logger.cpp File Reference

#include <Logger.h>
Include dependency graph for Logger.cpp:



# **Macros**

- #define  $\mathbf{cbi}(\mathsf{sfr}, \mathsf{bit})$  (\_SFR\_BYTE( $\mathsf{sfr})$  &=  $\sim$ \_BV( $\mathsf{bit})$ )
- #define **sbi**(sfr, bit) (\_SFR\_BYTE(sfr) |= \_BV(bit))

# **Functions**

- void wakeUpNow ()
- void wakeUpNow\_tip ()
- datafile **print** (T, 4)
- datafile print (F(","))
- void \_ISR\_void ()
- void \_anemometer\_count\_increment ()

32 File Documentation

#### **Variables**

- const int bottle\_logger =0
- const int big log =1
- const int log\_mega =2
- const int **SCKpin** = 13
- const int MISOpin = 12
- const int MOSIpin = 11
- const int CSpin = 10
- const int **SDApin** = A4
- const int **SCLpin** = A5
- const int **SensorPowerPin** = 4
- const int SDpowerPin = 8
- const int ClockPowerPin = 6
- const int **LEDpin** = 9
- const int wakePin = 2
- const int interruptNum = wakePin-2
- const int manualWakePin = 5
- int dayInterval
- · int hourInterval
- int minInterval
- · int secInterval
- int \_days
- int \_hours
- int minutes
- int seconds
- bool \_use\_sleep\_mode = true
- bool CAMERA IS ON = false
- bool IS LOGGING = false
- char \* filename
- char \* logger\_name
- bool extInt
- bool NEW RAIN BUCKET TIP = false
- bool LOG\_ALL\_SENSORS\_ON\_BUCKET\_TIP
- unsigned int rotation\_count = 0
- RTClib RTC
- DS3231 Clock
- · SdFat sd
- · SdFile datafile
- SdFile otherfile
- DateTime now
- float **T0** = T0degC + 273.15
- float **Rinf** = R0\*exp(-B/T0)
- float **T** = B / log(Rtherm/Rinf)

### 6.1.1 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, 2011-2016, and Chad Sandell, 2016 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

Logger.cpp is part of Logger, an Arduino library written by Andrew D. Wickert and Chad T. Sandell Copyright (C) 2011-2015, Andrew D. Wickert Copyright (C) 2016, Andrew D. Wickert and Chad T. Sandell

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/.

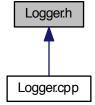
# 6.2 Logger.h File Reference

```
#include <Arduino.h>
#include <SoftwareSerial.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 dependency graph for Logger.h:



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



34 File Documentation

#### Classes

· class Logger

#### **Functions**

- void wakeUpNow ()
- void wakeUpNow tip ()
- · void ISR void ()
- void \_anemometer\_count\_increment ()

#### 6.2.1 Detailed Description

## Logger.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, 2011-2016, and Chad Sandell, 2016 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

Logger.h is part of Logger, an Arduino library written by Andrew D. Wickert and Chad T. Sandell.

Copyright (C) 2011-2015, Andrew D. Wickert Copyright (C) 2016, Andrew D. Wickert and Chad T. Sandell

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/.