**Atlas Scientific sensor running on Mayfly data logger**

**Overview**

A low cost Electrical Conductivity (EC) sensor using the Mayfly platform was developed in part to support Great Marsh Institute’s (GMI) efforts to study the effects of storm water runoff into Great Marsh. The main focus is road salt runoff from Rt 401 and the Pa Turnpike which border the marsh on the East and West sides.

Six monitoring sites are planned of which 3 are currently installed. This data will supplement EC data from DRWI sensor stations SL149 and SL150 which are deployed at Moores Rd and Fairfield road on Marsh Creek.

**Mayfly EC sensor details**

Since EC is temperature dependent a temperature reading is necessary so readings can be corrected to 25º C. A 10KΩ RTD is used to measure temperature which is then used to correct the reading to the reference temperature. [Sensorex](https://sensorex.com/product/cs150tc-tds-conductivity-probe/) provides a probe which has an RTD integrated with the probe. This eliminates the need for a separate sensor which would have to be mounted with the probe and in contact with the water.

The probe signal is processed by the Atlas [Scientific EZO circuit](https://www.atlas-scientific.com/_files/_datasheets/_circuit/EC_EZO_Datasheet.pdf). This circuit provides calibration routines for the attached EC probe but does not correct for temperature. This is done in the Mayfly code using a 2nd order polynomial fit to the data on the test fluid bottle.

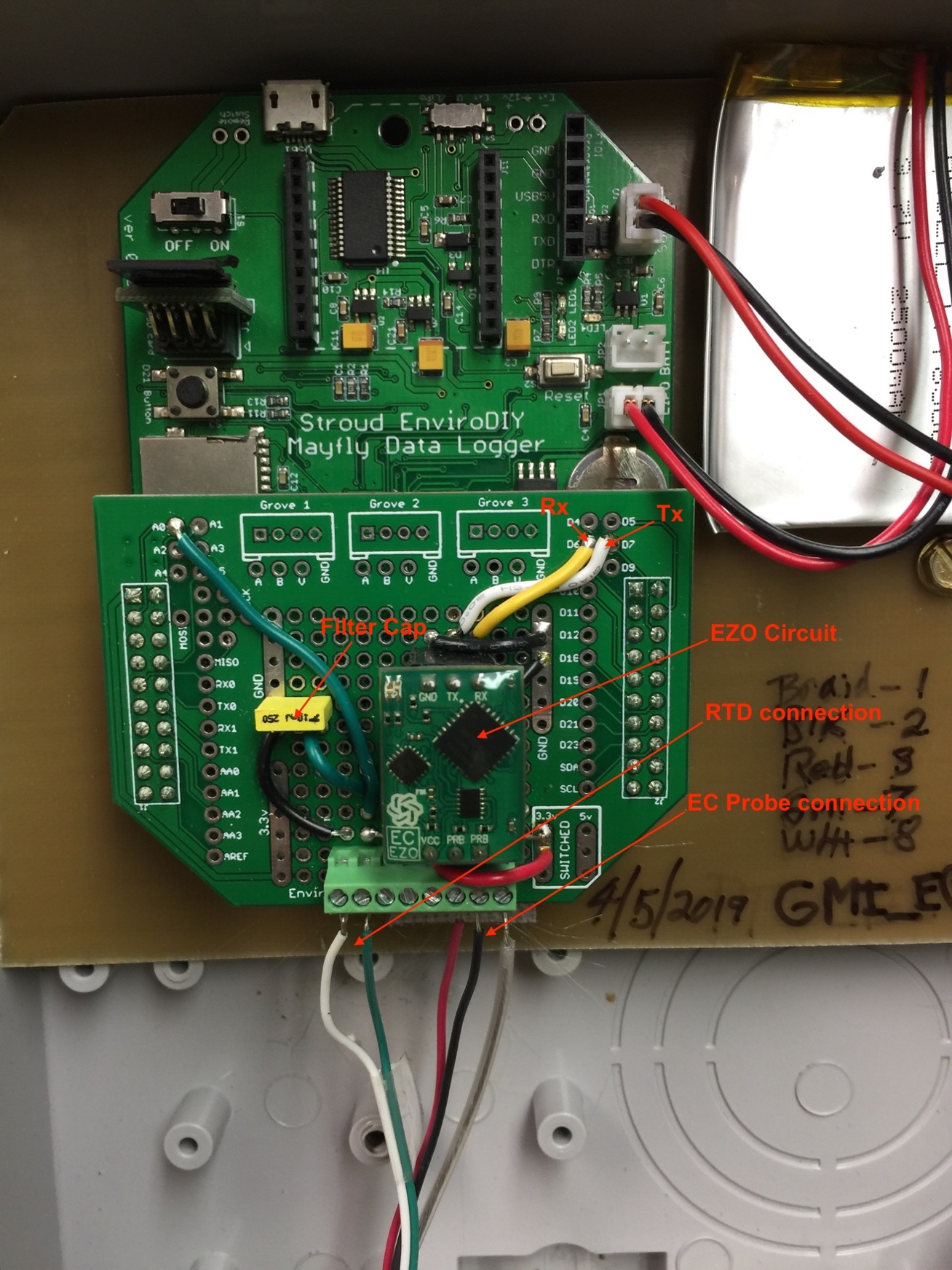
Prior to deployment the EZO Circuit needs to be configured. This includes setting up the EZO to send continuous readings at predetermined intervals and performing a probe calibration. The probe calibration can be done as single point or two point calibration depending on what calibration fluids are available. Single point calibration should be adequate using 1410 µS cal fluid.

**Hardware**

The following is a list of hardware needed with links to vendors and approximate price:

1. [Sensorex EC probe with RTD](https://sensorex.com/product/cs150tc-tds-conductivity-probe/?hsCtaTracking=e50e21b4-9e31-4c5a-b8ff-828319dd20a0%7Cf8427635-3f75-4579-b7b8-f7d7bbee0243) - $140
2. [Mayfly - $60](https://www.amazon.com/gp/product/B01F9B4WCG/ref=ppx_yo_dt_b_asin_image_o03_s03?ie=UTF8&psc=1)
3. [Atlas EZO -$60](https://www.amazon.com/Atlas-Scientific-EZO-EC-Embedded-Conductivity/dp/B006ERPKLM/ref=pd_rhf_ee_p_img_11?_encoding=UTF8&psc=1&refRID=QQD8AZEBRC17G0GBNXRZ)
4. [Cable Box](https://www.ebay.com/i/222032888777?chn=ps&norover=1&mkevt=1&mkrid=711-117182-37290-0&mkcid=2&itemid=222032888777&targetid=474173459509&device=c&mktype=pla&googleloc=9007350&poi=&campaignid=1669934834&mkgroupid=65058350339&rlsatarget=pla-474173459509&abcId=1139296&merchantid=101980317&gclid=EAIaIQobChMIk82SybfC4wIVEFYMCh03yg5REAQYASABEgJaPvD_BwE) - $12
5. [ProtoShield - $10](https://www.amazon.com/gp/product/B06Y5NP42G/ref=ppx_yo_dt_b_asin_image_o03_s00?ie=UTF8&psc=1)
   1. 10KΩ resistor
   2. 0.01µF Capacitor
   3. 6 pos [mini screw terminal](https://www.digikey.com/product-detail/en/on-shore-technology-inc/OSTVN06A150/ED10565-ND/1588866) - $2
   4. Socket strip for EZO
6. [2500 mAh](https://www.adafruit.com/product/328) Li-Ion battery - $15
7. [1 Watt solar panel](https://www.adafruit.com/product/3809) - $20

Total cost: $319



Mayfly circuit board showing ProtoShield with EC componants. The leads to the mini terminal strip go to the Sensorex EC Probe with 10K RTD

**Basic block diagram:**

EC Probe

10KΩ RTD

Tx/Rx serial

EZO

Sensorex probe

Mayfly Board

RTD%20ckt.pdf

RTD Circuit



EC Sensor ready to deploy –PVC assembly slipped over ½” pipe driven into stream bed.

**Mayfly Code**

Initially the Atlas/Mayfly sensor needs to have the Atlas EZO circuit setup. This includes probe calibration and setting the operational mode to continuous (“C”) output rather than triggered (“R”). Refer to the [Atlas EZO datasheet](https://www.atlas-scientific.com/_files/_datasheets/_circuit/EC_EZO_Datasheet.pdf) for details. Once this is done the operational code can be loaded which reads the EC and temperature data, corrects the EC reading to 25ºC reference and stores the data along with a time stamp on a µSD card.

When the station wakes from a predefined sleep period the EZO is powered up and begins to take readings at two second (recommended) intervals. It takes several readings for the results to stabilize so typically 10 readings are taken and the last reading is logged to the SD card. The station then goes to sleep for a specified time period.

**Note:** The code uses the D10 hardware interrupt therefor SJ1 must be cut from A7 and connected to D10.

**Mayfly Setup Proceedure:**

1. Set RTC to local standard time for consistency. Note that opening the serial monitor will reset the time unless the *rtc.setDateTime(dt)* is commented out.
2. Load the “Atlas EZO setup” code to calibrate the EC probe per the Atlas EZO datasheet. A two point calibration is recommended using 84µS and 1413µS cal fluids. Remember to use the value of the fluid at the ambient temperature of the fluid when doing the calibration procedure. For example if the ambient temperature of the cal fluid is 20ºC then “cal,high, 1278” should be entered.
3. The RTD in the Sensorex probe seems to have fairly consistent calibration using the m and b coefficients in the code (*float m = -.0927, b = 72.653;*). The accuracy can be checked at two points; typically, an ice bath at 0ºC and warm water at about 40ºC with errors used to correct the coefficients.
4. Before loading the “EC Monitor” code be sure to set the EZO to continuous update of 2 seconds (“c,2”). It was found that with a 1 second update some devices would produce a zero reading for unknown reasons. However, when the update was set to two seconds the problem did not reoccur.
5. Load the “EC monitor” code with the values of *sleepMinutes, FILE\_NAME,* and *LOGGERNAME* set to user preference.

**Set RTC code**

// Date and time functions using a RX8025 RTC connected via I2C and Wire lib

#include <Wire.h>

#include "Sodaq\_DS3231.h"

char weekDay[][4] = {"Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat" };

//year, month, date, hour, min, sec and week-day(starts from 0 and goes to 6)

//writing any non-existent time-data may interfere with normal operation of the RTC.

//Take care of week-day also.

DateTime dt(2019, 8, 13, 23, 31 , 0, 2);

void setup ()

{

Serial.begin(9600);

Wire.begin();

rtc.begin();

//Allow 8 sec to upload so as to match dt. Do not open serial monitor

//untill code reloaded with the next line commented out

//rtc.setDateTime(dt); //Adjust date-time as defined 'dt' above

}

void loop ()

{

DateTime now = rtc.now(); //get the current date-time

Serial.print(now.year(), DEC);

Serial.print('/');

Serial.print(now.month(), DEC);

Serial.print('/');

Serial.print(now.date(), DEC);

Serial.print(' ');

Serial.print(now.hour(), DEC);

Serial.print(':');

Serial.print(now.minute(), DEC);

Serial.print(':');

Serial.print(now.second(), DEC);

Serial.println();

Serial.print(weekDay[now.dayOfWeek()]);

Serial.println();

delay(1000);

}

**Atlas EZO setup code**

/\*

3/16/19 Modified getTemp()for 10K RTD

\*/

#include <SoftwareSerial.h> //we have to include the SoftwareSerial library, or else we can't use it.

#include <SD.h>

#include <RTCTimer.h>

#include <Sodaq\_DS3231.h>

#define rx 7 //D7 to Tx on EZO (white on Grove conn)

#define tx 6 //D6 to Rx on EZO (yellow on Grove conn)

#define SD\_SS\_PIN 12 //orig had pin 11

//The data log file

#define FILE\_NAME "datafile.txt"

//Data header

#define LOGGERNAME "Atlas Sci Sensor"

#define DATA\_HEADER "DateTime\_EST,Loggertime,BoardTemp\_C,Battery\_V,ECTemp,EC"

SoftwareSerial myserial(rx, tx); //define how the soft serial port is going to work.

String inputstring = ""; //a string to hold incoming data from the PC

String sensorstring = ""; //a string to hold the data from the Atlas Scientific product

boolean input\_stringcomplete = false; //have we received all the data from the PC

boolean sensor\_stringcomplete = false; //have we received all the data from the Atlas Scientific product

//used to hold a floating point number that is the pH.

float Temp0 = 0;

float m = -.0927, b = 72.653; //Calibration for RTD on Sensorex probe

long currentepochtime = 0;

int currentminute;

String dataRec = "";

float boardtemp;

int batteryPin = A6; // to read battery voltage

int batterysenseValue = 0; // variable to store the value coming from the sensor

float batteryvoltage;

void setup() { //set up the hardware

Serial.begin(9600); //set baud rate for the hardware serial port\_0 to 9600

myserial.begin(9600); //set baud rate for software serial port\_3 to 9600

inputstring.reserve(10); //set aside some bytes for receiving data from the PC

sensorstring.reserve(30); //set aside some bytes for receiving data from Atlas Scientific product

setupLogFile();

pinMode(22, OUTPUT); //Power to Grove connectors

digitalWrite(22, true); //turn power on

//pinMode(rx, INPUT);

}

void serialEvent() { //if the hardware serial port\_0 receives a char

char inchar = (char)Serial.read(); //get the char we just received

inputstring += inchar; //add it to the inputString

if (inchar == '\r') {

input\_stringcomplete = true; //if the incoming character is a <CR>, set the flag

}

}

void loop() { //here we go...

if (input\_stringcomplete) { //if a string from the PC has been received in its entirety

myserial.print(inputstring); //send that string to the Atlas Scientific product

inputstring = ""; //clear the string

input\_stringcomplete = false; //reset the flag used to tell if we have received a completed string from the PC

}

if (myserial.available() > 0) { //if we see that the Atlas Scientific product has sent a character.

char inchar = (char)myserial.read(); //get the char we just received

sensorstring += inchar;

if (inchar == '\r') {

sensor\_stringcomplete = true; //if the incoming character is a <CR>, set the flag

}

}

if (sensor\_stringcomplete) { //if a string from the Atlas Scientific product has been received in its entirety

recieveSensorData();

}

} //end of loop

void setupLogFile()

{

//Initialise the SD card

if (!SD.begin(SD\_SS\_PIN))

{

Serial.println("Error: SD card failed to initialise or is missing.");

//Hang

// while (true);

}

//Check if the file already exists

bool oldFile = SD.exists(FILE\_NAME);

//Open the file in write mode

File logFile = SD.open(FILE\_NAME, FILE\_WRITE);

//Add header information if the file did not already exist

if (!oldFile)

{

logFile.println(LOGGERNAME);

logFile.println(DATA\_HEADER);

}

//Close the file to save it

logFile.close();

}

void logData(String rec)

{

//Re-open the file

File logFile = SD.open(FILE\_NAME, FILE\_WRITE);

//Write the CSV data

logFile.println(rec);

//Close the file to save it

logFile.close();

}

String getDateTime()

{

String dateTimeStr;

//Create a DateTime object from the current time

DateTime dt(rtc.makeDateTime(rtc.now().getEpoch()));

currentepochtime = (dt.get()); //Unix time in seconds

currentminute = (dt.minute());

//Convert it to a String

dt.addToString(dateTimeStr);

return dateTimeStr;

}

void recieveSensorData()

{

if (sensor\_stringcomplete) { //if a string from the EZO product has been received

//in its entirety

dataRec = createDataRecord();

logData(dataRec); //Save the data record to the log file

}

Serial.print(" ");

Serial.println(dataRec);

sensorstring = ""; //clear the string:

sensor\_stringcomplete = false; //reset the flag Rec to tell if we have received a

}

String createDataRecord()

{

//Create a String type data record in csv format

//TimeDate, Loggertime,Temp\_DS, Diff1, Diff2, boardtemp

String data = getDateTime();

data += ",";

rtc.convertTemperature(); //convert current temperature into registers

boardtemp = rtc.getTemperature(); //Read temperature sensor value

batterysenseValue = analogRead(batteryPin);

batteryvoltage = (3.3 / 1023.) \* 4.7 \* batterysenseValue; //4.7 for rev 0.5

data += currentepochtime;

data += ",";

addFloatToString(data, boardtemp, 3, 1); //float

data += ",";

addFloatToString(data, batteryvoltage, 4, 2);

data += " , "; //adds a comma between values

Temp0 = getTemp();

addFloatToString(data, Temp0, 4, 2); //float

data += ",";

data += sensorstring; //needs to be last record added to dataRecord

//due to built in cr/lf

return data;

}

float getTemp() {

float Temp;

Temp = analogRead(A0);

Temp = m \* Temp + b;

return Temp;

}

static void addFloatToString(String & str, float val, char width, unsigned char precision)

{

char buffer[10];

dtostrf(val, width, precision, buffer);

str += buffer;

}

**EC Monitor code**

/\*

3/26/19 added linear cal for RTD using 10k divider

\*/

#include <Wire.h>

#include <avr/sleep.h>

#include <avr/wdt.h>

#include <SPI.h>

#include <SD.h>

#include <SoftwareSerial.h>

#include <RTCTimer.h>

#include <Sodaq\_DS3231.h>

RTCTimer timer;

#define rx 7 //D7 to Tx on EZO (white on Grove conn)

#define tx 6 //D6 to Rx on EZO (yellow on Grove conn)

SoftwareSerial myserial(rx, tx); //define how the soft serial port is going to work.

String dataRec = "";

String sensorstring = "EZO data"; //a string to hold the data from the EZO product

float Temp0 = 0;

//float K\_temp = 0.034; //2/17/19 update with new LM35

float m = -.0927, b = 72.653; //Calibration for RTD on Sensorex probe

float EC, EC25;

float alpha; //Temperature correction to EC at 25C

float K0 = 0.018, K1 = 8E-5, K2 = 1E-6;

int currentminute;

int n = 0;

int nMax = 10; //number of readings to take on each wake period

int nSkip = 10; //readings to skip on startup so only valid data is logged

int sleepMinutes = 5; //total minutes the Mayfly is in sleep mode

long currentepochtime = 0;

float boardtemp;

int batteryPin = A6; // to read battery voltage

int batterysenseValue = 0; // variable to store the value coming from the sensor

float batteryvoltage;

boolean sensor\_stringcomplete = false; //have we received all the data from the EZO product

//RTC Interrupt pin

#define RTC\_PIN 10 //Was A7 for old board

#define RTC\_INT\_PERIOD EveryMinute

#define SD\_SS\_PIN 12 //orig had pin 11

//The data log file

#define FILE\_NAME "EC3data.txt"

//Data header

#define LOGGERNAME "GMI\_EC#3"

#define DATA\_HEADER "DateTime\_EST,Loggertime,BoardTemp\_C,Battery\_V,Temp\_C,EC\_25"

void setup()

{

//Initialise the serial connection

Serial.begin(9600);

myserial.begin(9600); //set baud rate for software serial port\_3 to 9600

//if myserial active sleep doesn’t work

rtc.begin();

pinMode(8, OUTPUT);

pinMode(9, OUTPUT);

pinMode(22, OUTPUT); //Power to Grove connectors

sensorstring.reserve(30); //set aside some bytes for receiving data from EZO product

greenred4flash(); //blink the LEDs to show the board is on

setupLogFile();

setupTimer(); //Setup timer events

setupSleep(); //Setup sleep mode

digitalWrite(tx, false); //Turn off tx to EZO ckt

digitalWrite(rx, false); //Turn off rx to EZO ckt

digitalWrite(22, true); //Turn on power to EZO ckt

Serial.println("Power On, running: mayfly\_sleepEZO.ino");

showTime(getNow());

}

void loop()

{

//Update the timer

timer.update();

if (currentminute % sleepMinutes == 0) //will wake every minute for set delay untill condition

//is satisfied

{

while (!sensor\_stringcomplete && (n < nMax))

{

recieveSensorData();

}

n = 0; //reset EZO read counter

}

delay(100); //should be less than sensor turnon time which is ~1sec

//Sleep

systemSleep();

} //end loop

void sensorsSleep()

{

Serial.println("..going to sleep!");

digitalWrite(22, false); //Turn off power to EZO ckt

digitalWrite(tx, false); //Turn off tx to EZO ckt

digitalWrite(rx, false); //Turn off rx to EZO ckt

delay(100); //adding delay fixed the never-sleep problem

}

void sensorsWake()

{

digitalWrite(22, true); //Turn on power to EZO ckt

Serial.println("..I'm awake!");

delay(100);

}

void recieveSensorData()

{

if (myserial.available() > 0) { //if we see that the EZO product has sent a character.

char inchar = (char)myserial.read(); //get the char we just received

sensorstring += inchar;

if (inchar == '\r') {

sensor\_stringcomplete = true; //if the incoming character is a <CR>, set the flag

n++;

}

}

if (sensor\_stringcomplete) { //if a string from the EZO product has been received

//in its entirety

dataRec = createDataRecord();

if (n >= nSkip) //don’t log sensor startup response

{

logData(dataRec); //Save the data record to the log file

}

Serial.print(" ");

Serial.println(dataRec);

Serial.print("n= ");

Serial.println(n);

sensorstring = ""; //clear the string:

sensor\_stringcomplete = false; //reset the flag used to tell if we have received a

}

}

String createDataRecord()

{

//Create a String type data record in csv format

//TimeDate, Loggertime,Temp\_DS, Diff1, Diff2, boardtemp

String data = getDateTime();

data += ",";

rtc.convertTemperature(); //convert current temperature into registers

boardtemp = rtc.getTemperature(); //Read temperature sensor value

batterysenseValue = analogRead(batteryPin);

batteryvoltage = (3.3 / 1023.) \* 4.7 \* batterysenseValue; //4.7 for rev 0.5

data += currentepochtime;

data += ",";

addFloatToString(data, boardtemp, 3, 1); //float

data += ",";

addFloatToString(data, batteryvoltage, 4, 2);

data += " , "; //adds a comma between values

Temp0 = getTemp();

addFloatToString(data, Temp0, 4, 2); //float

data += ",";

//correct EC to 25C reference using data from Hanna 1413uS standard

EC = sensorstring.toFloat();

alpha = K2 \* pow(Temp0, 2) + K1 \* Temp0 + K0;

EC25 = EC / (1 + alpha \* (Temp0 - 25));

addFloatToString(data, EC25, 4, 1);

//data += sensorstring; //needs to be last record added to dataRecord

//due to built in cr/lf

return data;

}

void showTime(uint32\_t ts)

{

//Retrieve and display the current date/time

String dateTime = getDateTime();

//Serial.println(dateTime);

}

void setupTimer()

{

//Schedule the wakeup every minute

timer.every(1, showTime);

//Instruct the RTCTimer how to get the current time reading

timer.setNowCallback(getNow);

}

void wakeISR()

{

//Leave this blank

}

void setupSleep()

{

pinMode(RTC\_PIN, INPUT\_PULLUP);

attachInterrupt(2, wakeISR, CHANGE); //this used "PcInt::" for old board

//Setup the RTC in interrupt mode

rtc.enableInterrupts(RTC\_INT\_PERIOD);

//Set the sleep mode

set\_sleep\_mode(SLEEP\_MODE\_PWR\_DOWN);

}

void systemSleep()

{

//This method handles any sensor specific sleep setup

sensorsSleep();

//Wait until the serial ports have finished transmitting

Serial.flush();

// Serial1.flush(); //why??

//The next timed interrupt will not be sent until this is cleared

rtc.clearINTStatus();

//Disable ADC

ADCSRA &= ~\_BV(ADEN);

//Sleep time

noInterrupts();

sleep\_enable();

interrupts();

sleep\_cpu();

sleep\_disable();

//Enbale ADC

ADCSRA |= \_BV(ADEN);

//This method handles any sensor specific wake setup

sensorsWake();

}

String getDateTime()

{

String dateTimeStr;

//Create a DateTime object from the current time

DateTime dt(rtc.makeDateTime(rtc.now().getEpoch()));

currentepochtime = (dt.get()); //Unix time in seconds

currentminute = (dt.minute());

//Convert it to a String

dt.addToString(dateTimeStr);

return dateTimeStr;

}

uint32\_t getNow()

{

currentepochtime = rtc.now().getEpoch();

return currentepochtime;

}

void greenred4flash()

{

for (int i = 1; i <= 4; i++) {

digitalWrite(8, HIGH);

digitalWrite(9, LOW);

delay(50);

digitalWrite(8, LOW);

digitalWrite(9, HIGH);

delay(50);

}

digitalWrite(9, LOW);

}

void setupLogFile()

{

//Initialise the SD card

if (!SD.begin(SD\_SS\_PIN))

{

Serial.println("Error: SD card failed to initialise or is missing.");

//Hang

// while (true);

}

//Check if the file already exists

bool oldFile = SD.exists(FILE\_NAME);

//Open the file in write mode

File logFile = SD.open(FILE\_NAME, FILE\_WRITE);

//Add header information if the file did not already exist

if (!oldFile)

{

logFile.println(LOGGERNAME);

logFile.println(DATA\_HEADER);

}

//Close the file to save it

logFile.close();

}

void logData(String rec)

{

//Re-open the file

File logFile = SD.open(FILE\_NAME, FILE\_WRITE);

//Write the CSV data

logFile.println(rec);

//Close the file to save it

logFile.close();

}

static void addFloatToString(String & str, float val, char width, unsigned char precision)

{

char buffer[10];

dtostrf(val, width, precision, buffer);

str += buffer;

}

float getTemp() {

float Temp;

Temp = analogRead(A0);

//Temp = Temp \* K\_temp; for LM35

Temp = m \* Temp + b;

return Temp;

}