Master Thesis Document

# Grand Plan Outline

Author: William Kerr

# Introduction:

Goals:

My goal is to create a solar-powered greenhouse. The requirements of the greenhouse are to store plants at a reasonable temperature, store a monitoring system inside, and power said monitoring system with rooftop solar panels providing power.

UCSC has 24 solar panels that provide 1.5kWhr of power per day. The power will be delivered to the system properly using a MPPT Solar Charge Controller, provided by Morningstar. Each charge controller can only take 150V, and the total amount of solar panels we have sums up to 196V. Therefore, we will have two charge controllers working in tandem to charge a battery pack. These charge controllers can charge the same battery pack without interference, according to Morningstar. Both charge controllers will have a remote temperature sensor connected to it in order to monitor the amount of charge being delivered to the batteries. The charge controllers will deliver power to a battery pack, but these charge controllers cannot regulate all the batteries by themselves.

The battery pack consists of 8 batteries, each with a maximum charge of 180Ah and nominal voltage of 3.2V, provided by CALB. The batteries will be connected in series in groups of four, summing up to 12.8V per group. The battery pack will be stabilized, its output current regulated, and its temperature regulated by a Battery Management System (BMS), provided by EMUS. The BMS consists of a central control module, two CAN Cell Group Modules, two top isolators, two bottom isolators, and 8 Cell Isolator boards. All components will be controlled by a CAN bus originating from the central control module, that EMUS has programmed for its end users.

The BMS will control the charging of the batteries using variables such as the voltage, current, temperature, and charge of each individual cell, and modify internal variables such as the battery balancing rate accordingly to prevent damage to the batteries. If an individual battery dies, the BMS system will alert you with its monitoring system. The monitoring system can monitor from each battery: current drawn, individual voltage, temperature, and more. The monitoring system will calculate how much charge the battery has based on its own internal logic system. If the charge of any battery gets too low, it will focus the incoming charge from the solar panels on that particular battery. If the maximum charge of any battery gets too low, it will let you know with an alert. The monitoring system will monitor the batteries’ temperature. The BMS will cool the batteries if they get too hot, and warm the batteries if they get too cold. The monitoring system will let you know with an alert. There are other variables the monitoring system monitors. Check the components document for details.

The battery pack will power the internal computer system inside the greenhouse, an AC outlet somewhere in the greenhouse, and the 12V distribution nodes scattered throughout the greenhouse. Battery power will be converted from DC to AC using a power converter called the Suresine inverter, provided by Morningstar, that will be located near the Charge Controller. The AC power outlet will have a maximum power output of 300W. The 12V distribution nodes will distribute power to various devices that one might need for a greenhouse experiment, such as an air pump, hanging lights, a heater, a water pump, etcetera.

The internal monitoring system consists of a Raspberry Pi main microcontroller, which will be located next to the Charge Controllers for convenience, a 2G GSM shield attached to the main microcontroller, a 7” touch screen for convenience, the faculty sensors, the student sensors, and the Raspberry Pi slave microcontroller(s).

The faculty sensors will need to monitor light, humidity, and temperature, as per the client request. This can be achieved with light sensors and humidity & temperature combination sensors. Since I have the liberty to choose which sensors I will use, I will choose sensors that utilize the I2C serial protocol. This way, a daughterboard can be designed that will enable the main microcontroller to monitor up to 8 I2C devices. The daughterboard will have detachable inputs, so it is possible to change the sensors utilized in the greenhouse.

The main Raspberry Pi microcontroller is powerful enough to suit our needs. It will be running a Linux system called Raspbian. I will install a touch screen display to the main microcontroller, so if a faculty member needs to debug it, they can see what they’re doing without needing to connect to the Raspberry Pi using an SSH connection. In case of an emergency, I will include a way to connect to the main Raspberry Pi using an SSH connection and the proper permissions. The main Raspberry Pi and the touch screen will have an enclosure to protect it from the elements. The main Raspberry Pi’s tasks will consist of reading the sensor values from the faculty sensors using the I2C serial protocol, receiving a student sensor value package from the slave computers located around the greenhouse over the Bluetooth protocol, reading and storing the values generated from the Charge Controller’s internal PLC, reading and storing the values generated from the Battery Management System’s internal logic system, compiling all of that data, and then sending it over a 2G TCP connection directly to a server somewhere at UCSC with a GSM/GPRS daughterboard attachment. The 2G connection will be provided by Ting, as per client request.

The student sensors can consist of almost any type of sensor, and it seems random at this point in time. Here are some possibilities I can envision: water temperature sensors, soil moisture sensors, light sensors, humidity & temperature combination sensors, pressure sensors. Since it is impossible to know what serial protocol the student sensors will utilize, I will incorporate the most popular serial protocols into a daughterboard I will design, including I2C, SPI, UART, and 1-wire. I will incorporate a lot of UART ports into my daughterboard design, so that way if their sensor uses a strange protocol not listed here, a faculty member or another engineering student can make a bridge from that protocol to UART, write some code to read their sensor, and the slave microcontroller will be able to include their sensor in the data packet it sends to the master microcontroller.

The slave Raspberry Pi microcontrollers will be powerful enough to send data packets to other microcontrollers using the Bluetooth protocol, read sensor values over a variety of serial protocols, and process that data and compile it into a data packet. The slave Raspberry Pi will have an enclosure to protect it from the elements, but it will not have a sensor. It will be programmable from the main Raspberry Pi. The data packets will contain this type of data: name of owner, name of sensor, type of sensor, protocol utilized, value itself, what unit it’s measured in, time and date of when the value was taken.

The main Raspberry Pi microcontroller will be sending a packet of data, utilizing the GSM daughterboard, to a UCSC server somewhere. Once the server receives the data, it will store it in a SQL-style database. That server will also be connected to the internet, so users can connect to it from their phones. When a user connects to this server, the server will serve a website back to the user. The user can then use that website to send a request to the server asking for student sensor data, faculty sensor data, Battery Management data, or Charge Controller data. The user can add a format to their request, such as a graph over time, instantaneous data, data from the last time the computer has polled, data from a specific point in time, etc.

If the user requests a graph over time, they will need to specify what type of data they are requesting, Y-Axis units (i.e. V, mA, etc.), X-axis data (i.e. units of time), and range and scale for both X and Y axis for each dataset they request. Optional: a caption, a title, names for the X and Y axis. The server will return a webpage containing their graph, and some graph metadata: how many data points are returned, what the scale and range are, and a legend if they requested multiple data sets.

# Sources:

The Charge controller can work in tandem with another charge controller of the same brand:

<https://www.morningstarcorp.com/parallel-charging-using-multiple-controllers-separate-pv-arrays/>

**List of Components**

***Author: William Kerr***

# Solar Panels:

## Introduction:

This is a client request.

These solar panels let certain wavelengths of light through them, and absorb the rest of the spectrum.

This allows plants to grow inside.

Model: LUMO 20M100GH

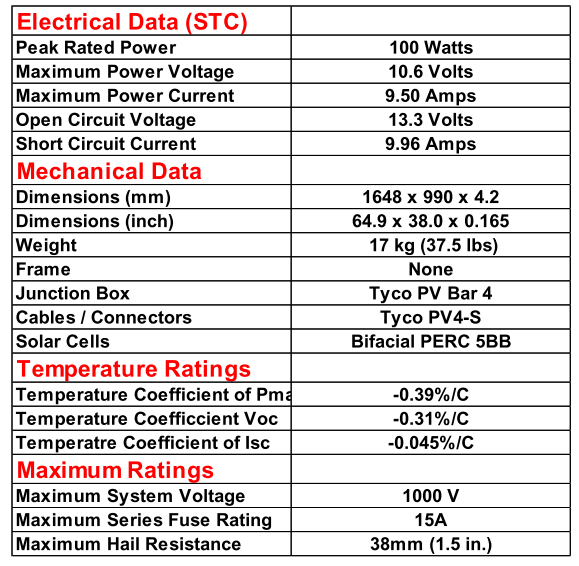
Quantity: 24x

Company: Soliculture

Company Website: <http://www.soliculture.com/>

Product Page: <http://www.soliculture.com/product/>

## Datasheet:



# Charge Controller:

## Introduction:

Solar panels cannot charge batteries directly for these reasons:

1.) They have unstable voltages, and thus should not be connected directly to the battery.

2.) Batteries with different chemical compositions charge differently.

3.) One solar panel cannot provide enough power to the battery alone, even if it reaches the nominal voltage of 12V.

4.) If we string 12 solar panels in series together and plug it into the batteries, the batteries will become permanently damaged.

Solar panels cannot charge batteries properly by themselves. We must have a charge controller to accompany the solar panels in order to charge the batteries properly. I like the TS-MPPT-60 because it is custom-programmable, and it can monitor a significant amount of values that could be useful someday. I included the values it can monitor below.

Model: MORNINGSTAR TS-MPPT-60 TriStar MPPT 150V

Company: Morningstar

Company Website: [https://www.morningstarcorp.com](https://www.morningstarcorp.com/)

Product link: <https://www.morningstarcorp.com/products/tristar-mppt/>

Quantity: 2x

## Features:

Customizable Charge Settings

Great networking capabilities

RS-232 electrical interface for Microcontroller communication.

Uses royalty-free MODBUS protocol for easy data harvesting

Operating Range: -40C to 40C

Up to 60A continuous battery current

Compatible with 12V, 24V, and 48V battery systems

Maximum 150V solar panels in series

Keyholes for mounting

Uses TrakStar MPPT technology to track the maximum power point of the solar panels.

Temperature compensation

Two Tristar Morningstar MPPT’s can be attached to the same battery pack

## Drawbacks:

The internal PLC settings can only be changed with a PC that can run MSView, Morningstar’s proprietary program that can program any Morningstar device, that can be downloaded from Morningstar’s website: <https://www.morningstarcorp.com/msview/>

It can only handle 150V of solar panels. We have 24 solar panels, which in series total to a nominal voltage of 192V. Therefore, we must have at least 2.

## Monitoring:

Tristar Morningstar MPPT can monitor:

### Internal ADC chips:

Battery Voltage

Battery Terminal Voltage

Battery Sense Voltage

Array Voltage (of the solar panels)

Battery Current

Array Current (of the solar panels),

12V supply,

3V supply,

meterbus voltage,

1.8V supply,

Reference voltage

### Temperature Data:

Heatsink Temperature

RTS temperature

Battery Regulation Temperature

### Status Data:

Battery Voltage (slow)

Charging Current (slow)

Minimum Battery Voltage

Maximum Battery Voltage

Hourmeter

Faults raised

Alarms raised

LED state

DIP switch status

### MPPT Data:

Output Power

Input Power

Max power of last sweep

Vmp of last sweep

Voc of last sweep

### Charger Data:

Charge state

Target Regulation Voltage

Ah charge resettable:

Ah charge total

kWhr charge resettable

kWhr charge total

### Daily Data:

Battery Voltage Minimum

Battery Voltage Maximum

Input Voltage Maximum

Amp Hours accumulated

Watt hours accumulated

Minimum Power output

Minimum temperature

Maximum temperature

Time in ab stage

Time in equalize stage

Time in float stage

Alarms of the day

Faults of the day

Flags of the day

### Current Charge Settings:

EV\_absorp

EV\_float

Et\_absorp

Et\_absorp\_ext

EV\_absorp\_ext

EV\_float\_cancel

Et\_float\_exit\_cum

EV\_eq

Et\_eqcalendar

Et\_eq\_above

Et\_eq\_reg

Et\_battery\_service

EV\_tempcomp

EV\_hvd

EV\_hvr

Evb\_ref\_lim

ETb\_max

Etb\_min

Elb\_lim

EVa\_ref\_fixed\_init

EVa\_ref\_fixed\_pet\_init

LED settings

EV\_soc\_g\_gy

EV\_soc\_gy\_y

EV\_soc\_y\_yr

EV\_soc\_yr\_r

## Recommended Accessories:

### Remote Temperature Sensor

Introduction:

The greenhouse will naturally change temperature more than 5 C during the year. The Morningstar Corporation recommends that you add the RTS sensor for the Charge Controller to operate more effectively under these circumstances. It is simple to install. Follow Morningstar’s guide to installation.

Model: Remote Temperature Sensor

Quantity: 2x

Company: Morningstar

Company Website: <https://www.morningstarcorp.com/>

Product Page: <https://www.morningstarcorp.com/products/remote-temperature-sensor/>

### RS-232 to USB cable

Introduction:

RS-232 must be converted to USB format for easy monitoring by the Raspberry Pi. Luckily, I don’t have to reinvent the wheel. I can simply use this cable. It has an FTDI chip and board embedded inside the plug, so I don’t have to worry about fabricating a chip.

I’m going with a USB terminal because my microcontroller is a Raspberry Pi, and it’s simpler to use a USB and create a virtual COM port inside the Raspberry Pi’s Linux operating system.

RS-232 Male to USB male

Model: C2G 26886 USB to DB9 Serial RS232 Adapter Cable, Blue (1.5 Feet, 0.45 Meters)

Quantity: 2x

Company: C2G

Company Website: <https://www.cablestogo.com/>

Product Page: <https://www.amazon.com/C2G-Cables-26886-Serial-Adapter/dp/B000067RVJ>

# Batteries:

Lithium-ion Battery Cell

## Introduction:

Solar panels do not produce power all the time. Even when they do produce power, they often don’t produce enough power to satisfy the consumer. During the day, when the solar panels produce the most power, the consumer often isn’t using the system. To resolve this, we need to have a battery pack. During the day, the battery pack will be charged by the solar panels, and during the evening, the battery pack will be discharged by the consumer.

Model: IFP71/180/278-CA180FI

Quantity: 8

Company: CALB

Company website: <http://www.calbusainc.com/>

Product Page: <https://www.ev-power.eu/LiFePO4-small-cells/Prismatic/CALB-CA180FI-Lithium-Cell-LiFePO4-3-2V-180Ah.html>

## Datasheet:

# Battery Management System:

## Introduction:

Batteries don’t discharge evenly. Every battery has its own individual chemistry due to imperfections in the manufacturing process. If we discharge batteries unevenly, one battery could be worn out while another battery remains untouched. To resolve this, we use a Battery Management System.

## Main Controller:

Model: G1 EMUS BMS control unit

Quantity: 1

Company: Emus

Company Website: <https://emusbms.com/>

Product Page: <https://emusbms.com/product/g1-bms-control-unit>

### Features:

Automatically controls the battery operation process utilizing various interfaces for measurement, control, data exchange, configuration and indication.

Works with any charge controller

Application:

Any lithium chemistry, series connected battery pack of up to 254 cells if using serial cell communication

Any lithium chemistry, series connected battery pack, or pack of multiple parallel strings, up to 8128 cells total, if using EMUS CAN Cell Group Modules.

Storage Temperature: -40 C to 95 C

Operation Temperature: -40 C to 80 C

USB interface for Microcontroller reading

Proprietary serial interface for cell communication

### Monitoring:

BMS control unit can monitor:

#### System Status

Battery Charge

Charger Status

Current and Voltage

Distance and Energy (if applied to an electric vehicle)

BMS status

Time and Date

Version Number

#### System status and Individual Cells:

Battery Balancing Rate

Temperature

Battery Voltage

#### Statistics

Has an internal events log (each event happening at a recorded time)

Has a statistics log at a recorded time. Possible statistics to log:

- Total Discharge

- Total Charge

- Total Discharge Energy

- Total Charge Energy

- Total Discharge Time

- Total Charge Time

- Total Distance

- Max Discharge Current

- Max Charge Current

- Min Cell Voltage

- Max Cell Voltage

- Max cell Voltage Difference

- Min pack voltage

- Max pack voltage

- Min Cell Module Temperature

- Max Cell Module Temperature

- Max Cell Module Temperature Difference

- Protection Counts (undervoltage, overvoltage, discharge overcurrent, charge overcurrent, cell module overheat, leakage protection, no cell communication, low voltage power reduction, high current power reduction, high cell module temperature power reduction, charger connect, charger disconnect, cell overheat, high cell module temperature power reduction)

- Miscellaneous counts (# of Preheat stages, Precharge stages, main charge stages, balancing stages, charging finished stages, charging errors, charging retries, trips, charge restarts)

- Min Cell Temperature

- Max Cell Temperature

- Max Cell Temperature Difference

## **Necessary Accessories:**

### Cell Isolators

The BMS system requires that you have isolators to protect the main module.

Only works if only 1 group of batteries is used.

Model: G1 Top/Bottom Isolator

Company Website: <https://emusbms.com/>

Product Page: <https://emusbms.com/product/g1-top-bot-isolator>

Quantity: 2x

### Cell Modules

Every battery must have its own cell module.

Different batteries require different cell modules.

You can find all types of cell modules here:

<https://emusbms.com/product-category/cell_modules>

The standard solution is the A/B type, so that’s what we’re going with.

We must order this package for each battery.

EMUS BMS Cell Module A – 1x

EMUS BMS Cell Module B – 1x

Ring Terminal M8 – 2x

Communication Cable – 16cm – 2x

#### Ordering details:

Model: G1 Cell Module – A/B type

Company: Emus

Company Website: [https://emusbms.com](https://emusbms.com/)

Product Page: <https://emusbms.com/product/g1-cell-module-ab>

Quantity: 8x

### CAN Cell Group Module

We need to group batteries into groups.

Since the batteries we picked are 3.2V, we group batteries into groups of 4.

Model: G1 CAN Cell Group Module

Company: Emus

Company Website: <https://emusbms.com/>

Product Page: <https://emusbms.com/product/g1-can-cell-group-module>

Quantity: 2x

## **Recommended Accessories:**

### Current Sensor

In order to monitor current dispensing from the batteries to the load, you must have a current sensor. It’s not necessary for operation, but it’s recommended to have one. This one works using the hall effect, so it does not require contact with the wires; it only needs to have the wire running through its hole.

Model: G1 Loop Style Dual Range Current Sensor

Company: Emus

Company Website: <https://emusbms.com/>

Product Page: <https://emusbms.com/product/g1-loop-style-dual-range-current-sensor>

Quantity: 1x

# Sensors – Faculty

## Introduction:

The client wants their own sensors exclusive for faculty. They want to measure temperature, humidity, and light. I propose that we use these classes of sensors for this:

* Temperature and Humidity Sensor
* Light Sensor

The product page for the parts and their respective datasheets will be hosted by different companies. This is because it is easier to order a breakout board than it is to order the individual parts, order a custom PCB for the sensor, and solder the parts onto the board. Companies that sell breakout boards and companies that manufacture parts are separate from one another.

## **Temperature and Humidity Sensor:**

Model: BME280

Company: Bosch

Company Website: <https://bosch.us/>

Product Page: <https://www.adafruit.com/product/2652>

Datasheet: <https://cdn-shop.adafruit.com/product-files/2652/2652.pdf>

### Details:

+-3% accuracy for humidity

+-1% accuracy for temperature

1s response time maximum

Operating range: -40C to 85C

I2C interface

Measures pressure if necessary

See datasheet for reading this sensor properly.

Create a class in C++/python to read it.

## **Light Sensor:**

Model: VEML7700

Quantity: 2

Company: Vishay Semiconductors:

Company Website:

Product Page: <https://www.adafruit.com/product/4162?gclid=EAIaIQobChMIyOmfve7Q4wIV6f_jBx07fQ1yEAQYASABEgJti_D_BwE>

Datasheet: <https://www.vishay.com/docs/84286/veml7700.pdf>

### Details:

High resolution: 0.0036 lux/ct at night, 1.8 lux/ct in bright sunlight

Maximum 120,000 lux (bright sunlight)

I2C interface

See datasheet for reading this sensor properly (i.e. what addresses to read from, what slave address to use, etc.)

Create a class in Python/C++ to read it.

# Faculty Microcontroller

## Introduction:

In every computer system, there must be a main processor. In this greenhouse system, a master microcontroller is utilized to harvest data, process it, and send it to a main server somewhere on campus. See the website manual for details on how it’s processed there. The main microcontroller must be able to communicate with the Faculty Sensors somehow, and communicate with the slave microcontrollers when we implement them. I have chosen to interface the main microcontroller with the slave microcontrollers via Bluetooth. Bluetooth is wireless, and easy to program. Does not require any wires running across the greenhouse floor, and reduces tripping hazard. So, our Microcontroller has these requirements:

* Must be capable of sending packets of data over a 2G internet connection to a server somewhere at UCSC.
* Must be capable of communicating over Bluetooth to a slave microcontroller somewhere in the greenhouse.
* Must be capable of reading I2C data.

The microcontroller I have chosen for this job is the Raspberry Pi Model 3B+. It is a capable microcontroller. It runs Linux on its systems, so it’s easy to debug on site if necessary. The code can be stored on an SD card. If necessary, it will be possible to retrieve a log of the past 30 days of data from the Raspberry Pi. The Raspberry Pi Raspbian system uses a FAT32 file system, meaning the absolute maximum amount of data it is possible of addressing is 32GB. So, it should be enough for at least 30 days worth of data. But, the SD card also has to store the operating system it will use (Raspian).

This microcontroller uses a +5V power source. Therefore, we will have to design a power source for it. The tolerance values for the microcontroller are tight: it only accepts +4.5V to +5.5V. It can draw up to 2A of current when running a stress-test. So, let’s just say it draws a maximum of 10W of power.

## Details:

Model: Raspberry Pi 3 Model B+

Quantity: 1

Company: Raspberry Pi

Company page: <https://www.raspberrypi.org/>

Product Page: <https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/>

## Features:

1.6GHz ARM processor

C++ compiler

Python interpreter

4 USB ports

20 GPIO pins

I2C, UART, and SPI interface

Runs Linux

Bluetooth and Wi-Fi Capabilities

Upgradeable

## Drawbacks:

Requires +4.5V to +5.5V of power.

Requires a Micro USB to power it. We can fabricate something that can deliver the necessary power to run it.

## Recommended Accessories:

### Sixfab’s GSM/GPRS shield:

I don’t like the FONA module. I would like to replace it. I would like to instead use this GSM/GPRS shield. It slides easily onto the master Raspberry Pi, and can also fit another shield onto it if so desired. I will be fabricating a faculty sensor shield utilizing the I2C protocol. This shield utilizes the UART protocol. The Raspberry Pi can only accommodate 1 use of the UART protocol using the GPIO pins. The others will be using the Virtual COM ports of the Raspberry Pi. The Tristars will be using a RS-232 to USB converters with an FTDI chip installed in them for communication, and the BMS system will be using a split-open USB wire that will connect directly to the BMS control unit.

Model: Raspberry Pi GSM/GPRS shield

Company: Sixfab

Company Page: <http://sixfab.com/>

Product Page: <https://sixfab.com/product/gsmgprs-shield/>

Features:

Uses Quectel M66 2G IoT modem.

Fully compatible with Raspberry Pi models that have the 40-pin GPIO header (3, 2, B+, A+, Zero)

High Data Speed: GPRS Multi-slot class 12, 85.6kbps downlink and 85.6kbps uplink data rates

Quad-band: 850/900/1800/1900MHz

Built-in PCB antenna, also there is an external antenna port available

Supported Protocols: TCP/ UDP/ PPP/ FTP/ HTTP/ SMTP/ CMUX/ SSL

Quectel’s QuecLocator Feature, lets you get the location without GPS/GNSS

Extremely low standby power consumption by M66, 1.3mA at DRX=5

Efficient and low quiescent current regulator circuit can hold up to 3.6A

Bluetooth Function, V3.0 specification, SPP and OPP profiles available.

Micro SIM Card socket can easily reachable on the downside of the shield.

Can be used standalone with PC/Laptop over micro USB, without stacking with Raspberry Pi thanks to FTDI chip on the shield.

Sending/Receiving standard V.25ter AT commands over UART port to Raspberry Pi is available

Working temperature range: -30°C to +80°C

### Antenna:

Any antenna that can physically connect to this shield will do. But, here’s one from Sixfab:

Model: GSM 2G/3G Antenna – u.FL PCB Antenna – 0dBi

Company: Sixfab

Company page: <https://sixfab.com/>

Product Page: <https://sixfab.com/product/gsm-2g-3g-antenna-u-fl-pcb-antenna-0dbi/>

### SIM Card:

If we will be sending data with our GSM module, we must have a SIM card to tell the cell phone tower what carrier we are using, and if we have permission to use their cell phone tower. The SIM card only stores 1 piece of data: our ID number. That’s all it does, but it’s very important.

Model: Ting GSM SIM card

Quantity: 1

Carrier: Ting

Company Website: <https://ting.com/>

Product Page: <https://ting.com/shop/gsmSIM>

You must register with Ting and pay a monthly fee of $50 for an unlimited 2G service plan.

### Custom-fabricated I2C shield for Raspberry Pi.

Will be custom-designed at home here at UCSC. Will be rushed, though. If I find a design, will be using it. Will have these features:

* Capable of holding at least 8 I2C devices
* Capable of detaching I2C devices at will, like a plug.
* Has Pull-up resistors embedded inside

## Touch screen for Raspberry Pi

This is not completely necessary, but it would be nice to be able to see what is happening inside the raspberry pi 3 at any given moment.

Model: Raspberry Pi Touch Display

Company: Raspberry Pi

Company Page: <https://www.raspberrypi.org/>

Product Page: <https://www.raspberrypi.org/products/raspberry-pi-touch-display/>

## Case for Raspberry Pi Touch Display

If we have a Raspberry Pi touch display, we will need a case for it to add that extra touch. We will need to find a way to mount it, though.

Model: RS Raspberry Pi 7-Inch LCD Touch Screen Case, Black, Model number FBA\_102035

Company: Raspberry Pi

Company Page: <https://raspberrypi.org/>

Product Page: <https://www.amazon.com/Raspberry-Pi-7-Inch-Touch-Screen/dp/B01GQFUWIC/ref=asc_df_B01GQFUWIC/?tag=hyprod-20&linkCode=df0&hvadid=309751315916&hvpos=1o1&hvnetw=g&hvrand=10505497938605347385&hvpone=&hvptwo=&hvqmt=&hvdev=c&hvdvcmdl=&hvlocint=&hvlocphy=9061320&hvtargid=pla-406360183578&psc=1&tag=&ref=&adgrpid=67183599252&hvpone=&hvptwo=&hvadid=309751315916&hvpos=1o1&hvnetw=g&hvrand=10505497938605347385&hvqmt=&hvdev=c&hvdvcmdl=&hvlocint=&hvlocphy=9061320&hvtargid=pla-406360183578>

# Sensors – Student

## Introduction

As an optional feature, the client would like the system to be capable of having students be able to use their own sensors. Here are some potentially useful sensors for students:

## Water Temperature sensors

### DS18B20

The DS18B20 is sold in a different form factor from ADAFRUIT. This form factor is more usable, so we will be using that one. The sensor and its datasheet are provided by Maxim Integrated.

Model: DS18B20

Quantity: 1

Company: Maxim Integrated

Company Website: <https://www.maximintegrated.com/en.html/>

Product Page: <https://www.adafruit.com/product/381?gclid=EAIaIQobChMIh5e--PmU4wIViJWzCh3vLA9XEAQYASABEgKZSfD_BwE/>

Datasheet: <https://datasheets.maximintegrated.com/en/ds/DS18B20.pdf>

Details:

Interface: One-wire

* Reduce Component Count with Integrated Temperature Sensor and EEPROM
* Unique 1-Wire® Interface Requires Only One Port Pin for Communication
* Measures Temperatures from -55°C to +125°C (-67°F to +257°F)
* ±0.5°C Accuracy from -10°C to +85°C
* Programmable Resolution from 9 Bits to 12 Bits
* No External Components Required
* Parasitic Power Mode Requires Only 2 Pins for Operation (DQ and GND)
* Simplifies Distributed Temperature-Sensing Applications with Multidrop Capability
* Each Device Has a Unique 64-Bit Serial Code Stored in On-Board ROM
* Flexible User-Definable Nonvolatile (NV) Alarm Settings with Alarm Search Command
* Identifies Devices with Temperatures Outside Programmed Limits
* Available in 8-Pin SO (150 mils), 8-Pin µSOP, and 3-Pin TO-92 Packages

## Soil Moisture Sensors:

This one is a potential keeper for students:

Model: I2C Soil Moisture Sensor

Company: White Boxes

Company Page: <https://www.whiteboxes.ch/>

Product Page: <https://www.whiteboxes.ch/shop/i2c-soil-moisture-sensor/?v=7516fd43adaa>

Features:

* Comes with its own Arduino and Raspberry Pi examples library, located here:
  + <https://github.com/Miceuz/i2c-moisture-sensor>
* Version 2.7.5
* Supply voltage 3.3V – 5V
* Current consumption: 1.1mA @ 5V, 0.7mA @ 3.3V when idle, 14mA @ 5V, 7.8mA @ 3.3V when taking a measurement. When constantly polling sensor at full speed, current consumption averages to 4.5mA @ 5V, 2.8mA @ 3.3V
* Operating temperature 0°C – 85°C
* Moisture reading drift with temperature – <10% over full temp range

## Pressure Sensors:

Honeywell manufactures this pressure sensor, but ADAFRUIT distributes it.

Model: Adafruit MPRLS Ported Pressure Sensor Breakout - 0 to 25 PSI

Company: Honeywell

Company Page: <https://sensing.honeywell.com/>

Product Page: <https://www.adafruit.com/product/3965?gclid=EAIaIQobChMIx4HYxr7-5AIVDtvACh3JSggcEAQYBSABEgJE7_D_BwE>

Datasheet: <https://sensing.honeywell.com/micropressure-mpr-series>

Comes with its own example code library: <https://github.com/adafruit/Adafruit_MPRLS>

# Student Microcontroller

## Introduction:

Instead of having every student plug into one microcontroller (which would require a lot of cables running around), I propose that for every experiment, we have a separate microcontroller that the student can take with them. The Raspberry Pi 3 Zero W is a great candidate for this. It’s Bluetooth enabled, so they aren’t burdened by a cable length. It’s just as powerful as the normal Raspberry Pi, with the addition of writing their own code for their own sensors.

We will have to use our own sensor shields.

## Raspberry Pi Zero W:

Model: Raspberry Pi 3 Zero W

Quantity: 2

Company: Raspberry Pi

Company Website: <https://raspberrypi.org/>

Product page: <https://www.adafruit.com/product/3400?gclid=EAIaIQobChMI9Lbyu_qU4wIVDp6fCh3MuA5QEAQYASABEgJT5PD_BwE/>

Details:

* 1GHz, single-core CPU
* 512MB RAM
* Mini HDMI and USB On-The-Go ports
* Micro USB power
* HAT-compatible 40-pin header
* Composite video and reset headers
* CSI camera connector

Networking:

* 802.11 b/g/n wireless LAN
* Bluetooth 4.1
* Bluetooth Low Energy (BLE)

Drawbacks:

* Requires a voltage range of +4.5V to +5.5V, and a power converter.
* Requires a Micro USB.

## Recommended Accessories:

### Custom SPI, I2C, UART, and One-Wire shield

Will be designed in-house. Everything will be documented. Requirements:

* Take these data formatting protocols: SPI, I2C, UART, 1-wire.
* Be a wall bug. It will plug in to a 12V power supply with a 12V converter.

### Custom Housing

Will be designed in-house. Everything will be documented. Requirements:

* Cover the Raspberry Pi Zero W from the elements.
* Have ports for:
  + The 5V Power supply
  + The SPI, I2C, UART, and 1-Wire
* Optional: Covers for the ports.

# Heater – Battery pack

## Introduction:

All battery packs must have a heater. When batteries get too cold, there is a possibility of permanent damage to the batteries, and an unnatural reduction of life cycles might occur. There is no heater for the battery pack currently on-site, but here is my proposal: install a heater for wherever the batteries are stored. The heater will be controlled by a relay, which will be controlled by the BMS system. The BMS system has temperature sensors on-board to tell when the batteries are getting too cold.

Model: Asixx Air Heater, 100W 12V Energy Saving PTC Car Fan Air Heater Constant Temperature Heating Element Heaters for Heater, Humidifier, Air Conditioning and More

Company: Asixx

Company Website:

Product Page: <https://www.amazon.com/Asixx-Constant-Temperature-Humidifier-Conditioning/dp/B07HCB95SJ/ref=asc_df_B07HCB95SJ/?tag=hyprod-20&linkCode=df0&hvadid=309851778232&hvpos=1o1&hvnetw=g&hvrand=4833336270821486334&hvpone=&hvptwo=&hvqmt=&hvdev=c&hvdvcmdl=&hvlocint=&hvlocphy=9061320&hvtargid=pla-574478162578&psc=1>

Details:

Rated Voltage: 12V

Rated Power: 100W

Mounting Hole Distance: Approx. 87mm / 3.4inch

Mounting Hole Size: Approx. 4mm / 0.2inch

Product Size: Approx. 6 \* 6 \* 4.2cm / 2.4 \* 2.4 \* 1.7inch

Product Weight: Approx. 120g

We will mount this wherever the battery pack is.

# Cooler – Battery pack

## Introduction:

We should be burying our batteries underground to protect against warmer temperatures (according to forum rumors). If the batteries have an excessive load, they become at risk for overheating. If the batteries are overheated for too long, they will become permanently damaged. We must have a cooler for our battery pack to cool the batteries. I propose we install a fan or a radiator. The air underground usually stays at least 25 – 30 C, so we can use that air to cool the batteries.

I don’t have a solid choice of cooler yet, but this is my best guess so far:

Model:

Company:

Company Page:

Product Page:

<https://www.amazon.com/Performance-Electric-Radiator-Mounting-%EF%BC%88Diameter/dp/B01N0686K5/ref=asc_df_B01N0686K5/?tag=hyprod-20&linkCode=df0&hvadid=241994092016&hvpos=1o4&hvnetw=g&hvrand=169244523548284450&hvpone=&hvptwo=&hvqmt=&hvdev=c&hvdvcmdl=&hvlocint=&hvlocphy=9061320&hvtargid=pla-449411562922&psc=1>

Details:

* Overall Diameter:11.73 INCH,
* Overall Thickness:2.56 INCH
* 1550 CFM Car Cooling Fans
* Amp Draw: 6.6 amp
* Watts:80W
* Blade Length: 11inch
* Number of Blades: 10
* Blade Type: straight
* High-quality, lightweight, and durable; great as replacement or upgrade from factory parts.
* Simple installation, no modifications required. Fan can be used as pusher or puller with the adaptable mounting kit.
* How to choose the correct fan?
  + Step 1: Confirm the width and depth of your radiator
  + Step 2: Compare the fan diameter and depth with radiator
  + Step 3: Choose the right size fan
  + ATTENTION: It can be installed only if the diameter of the fan is smaller than the width of radiator! Make sure the depth of the fan is smaller than the gap of radiator and other parts!

# AC outlet

Explanation:

Every once in a while, somebody will want to use an AC outlet to power a laptop or charge a phone. An AC outlet is absolutely necessary to do these things. To have an AC outlet on a DC power grid, we must have an inverter. Here’s our inverter:

Model: Morningstar Suresine Inverter 300W

Model number: SI-300-115V-UL (60Hz)

Company: Morningstar

Company Website: <https://www.morningstarcorp.com/>

Product Page: <https://www.morningstarcorp.com/products/suresine/>

**Details:**

Data Communications: RJ-11Connection with Morningstar Meterbus / MODBUS RTU (16-bit)

Continuous Power Rating: 300W @ 25 C

Peak Power Rating (10 minutes): 600W

DC input voltage: 10.0V - 15.5 V

Waveform: Pure Sine Waveform

AC Output Voltage (RMS): 220V or 115V +/- 10%

AC Output Voltage Frequency: 50 or 60 Hz +/- 0.1%

Peak efficiency: 92%

Total Harmonic Distortion (THD): < 4%

Self Consumption:

Inverter On (no load): 450mA

Inverter Off: 25mA

Stand-by: 55mA

Low voltage Disconnect (LVD): 11.5V or 10.5V

Low Voltage Reconnect; 12.6V or 11.6V

LVD Warning Threshold (buzzer): 11.8V or 10.8V

LVD Delay Period: 4 minutes

High voltage disconnect: 15.5V

High Voltage Reconnect: 14.5V

Standby On Threshold: ~8W

Standby Off Threshold: ~8W

High Temperature Disconnect: 95 C (heatsink)

High temperature reconnect; 80 C (heatisnk)

**Electronic Protections:**

Reverse Polarity (fused)

AC Short Circuit

AC overload

DC Terminals: Max wire size:

– 2.5 to 35 mm^2 / 14 to 2 AWG

Remote On/Off terminals: Max. Wire size:

– 0.25 to 1.0 mm^2 / 24 to 16 AWG

Enclosure: IP20

Cast anodized Aluminum

**Physical Characteristics:**

Dimensions: 213 x 152 x 105 mm (8.4 x 6.0 x 4.1 in)

Weight: 4.5 Kg/10.0lbs

AC terminals: Max wire size:

– 4 mm^2 / 12AWG

**Environmental Protections:**

Ambient Operating Temperature: -40 C to +45C

Storage Temperature: -55 C to +85C

Humidity: 100% (non-condensing)

Tropicalization: Conformal coating on PCBs. Epoxy encapsulated transformer and inductors.

# Accessories:

3A fuse

100A fuse

GFCI outlet:

Model: 15 Amp Self-Test smartlock pro slim duplex GFCI Outlet, white

Company: Home Depot

Company Website: <https://www.homedepot.com/>

Product Page: <https://www.homedepot.com/p/Leviton-15-Amp-Self-Test-SmartlockPro-Slim-Duplex-GFCI-Outlet-White-R02-GFNT1-0KW/206001533>

GFCI outlet box:

Model: 1-Gang Weather Box While-In-Use cover

Company: Home Depot

Company Website: [https://www.homedepot.com](https://www.homedepot.com/)

Product Page: [https://www.homedepot.com/p/1-Gang-Weather-Box-While-In-Use-Cover-WIU-1/206469236?cm\_mmc=Shopping%7CG%7CVF%7CD27E%7C27-6\_CONDUIT-BOXES-FITTINGS%7CNA%7CPLA%7c71700000033099037%7c58700003867178937%7c92700031086148565&gclid=EAIaIQobChMI2PLwrenQ4wIVAf\_jBx2q5Q92EAkYASABEgKGrPD\_BwE&gclsrc=aw.ds](https://www.homedepot.com/p/1-Gang-Weather-Box-While-In-Use-Cover-WIU-1/206469236?cm_mmc=Shopping|G|VF|D27E|27-6_CONDUIT-BOXES-FITTINGS|NA|PLA|71700000033099037|58700003867178937|92700031086148565&gclid=EAIaIQobChMI2PLwrenQ4wIVAf_jBx2q5Q92EAkYASABEgKGrPD_BwE&gclsrc=aw.ds)

Recommended Accessories:

RJ-11 Meterbus to USB MODBUS adapter

Model: Morningstar USB MeterBus Adapter > UMC-1

Company: Morningstar

Company Website: [https://www.morningstarcorp.com](https://www.morningstarcorp.com/)

Product Page: <https://solarflexion.com/umc-1?_vsrefdom=adwords&gclid=EAIaIQobChMIs6q9_eTQ4wIVef_jBx3u-AdIEAQYBSABEgKaCPD_BwE>

RJ-11 data communications cable

Model: USB Meterbus Adapter

Company: Morningstar

Company Website: <https://www.morningstarcorp.com/>

Product Page:<https://www.morningstarcorp.com/products/usb-meterbus-adapter/>

# Web Server

## Introduction:

We must have a server to send the data to. I’m sending all my data in JSON format so it’s easier for the website to read it.

In my down time (when I’m too tired to work on electronics), I would like to work on the website.

I would also like to use Squarespace temporarily. They advertise everywhere, they offer hosting, and they offer good templates.

I will be crossing the bridge when we get to it.

Also, I would like to call upon the help of a professor for this one.

Host: ucsc.edu

Website link: arboretum-backend.soe.ucsc.edu/ (user requested)

Server Location: ???

Who to call when things go bad: ???

Uptime Percentage: ???

Language Programmed in: ???

# Power Conversion

## Introduction:

All our microcontrollers that we are using need +5V to operate properly. Our batteries distribute power in +12V packages. Therefore, we need a way to convert that power reliably.

## Requirements:

* Tolerance ranges: +4.5V to +5.5V
* Ends in a Micro-USB plug

We can either design a DC-DC voltage converter ourselves, or we could try to get one from a commercial retailer and hope it has good enough tolerances for our Raspberry Pi. Either way, we must have a DC-DC voltage converter.

I propose that we get one from a commercial retailer. That way, if there are any malfunctions with it, we don’t have to build another one ourselves, we can simply buy another one from the same retailer.

Here’s my recommendation:

Model: DC-DC 12V to 5V 3A Micro USB Converter Voltage Step Down Regulator Waterproof Power Converters for Car Smartphone

Company: Car Power Technlogies (CPT)

Company Website:

Product Pages:

<https://www.amazon.com/Converter-Regulator-Waterproof-Converters-Smartphone/dp/B07H7X37T6/ref=asc_df_B07H7X37T6/?tag=hyprod-20&linkCode=df0&hvadid=241968535606&hvpos=1o1&hvnetw=g&hvrand=2561462318447783132&hvpone=&hvptwo=&hvqmt=&hvdev=c&hvdvcmdl=&hvlocint=&hvlocphy=9061320&hvtargid=pla-600746528101&psc=1>

<https://www.aliexpress.com/item/32581610768.html>

Sales pitch:

* This voltage converter uses high quality industrial-grade chip, with high conversion rate and stability
* Widely used for car stereo, radio, monitoring, LED display, electric fan, motors and other electrical appliances, etc.
* It supports 3 intelligent security protection including over-current protection, over temperature protection and output short circuit protection
* Copper wire has great electrical conductivity, low thermal, low carbon and save more power
* Fire retardant plastic casing and organic silicone sealing technology has strong thermal conductivity, IP67 ingress rating, waterproof and shockproof
* Type: Non-isolated step-down voltage converter

Features:

* Output Connector: Micro USB
* Input Voltage: DC12V
* Output Voltage: DC5V
* Output Current: 3A, 2.5A(no need to enhance heat dissipation for long time use)
* Output Power: 15W
* Max. Efficiency: >95%
* Operating Temperature: -40¡æ to 80¡æ
* Overall Length: 50cm/19.7in
* Block Size: 6.4\*2.7\*1.4cm/2.5\*1.1\*0.6in
* Weight: 41g(approx.)
* Working temperature: -40C to 80C

I propose we use this power converter for all our 5V devices.

Morningstar Monitoring Code Design Document

# Morningstar.py Design Document

Author: William Kerr

# Introduction:

The TSMPPT has a Programmable Logic Controller (PLC) inside. The PLC can be reprogrammed to respond to different battery voltages. Example: when the battery goes to 3.7V, the battery goes to the equalize stage. To reprogram the Tristar MPPT, you must use a PC and a RS-232 to USB cable, and the program MSView, provided by Morningstar. Link: <https://www.morningstarcorp.com/msview/>. I also have a document called Viewing Data on Morningstar Devices that explains how to use the software.

However, the Tristar MPPT can be monitored by any device capable of serial monitoring, such asa a Raspberry Pi. Morningstar.py will contain the code necessary to monitor data from a Tristar Morningstar MPPT solar charge controller.

Morningstar has a MODBUS specification document for the Tristar Morningstar MPPT. It should be in my App Notes section. If not, here’s a link:

Link: <https://www.stellavolta.com/content/MSCTSModbusCommunication.pdf>

Morningstar.py contains a class that reads PLC data, denoted as Morningstar(). This class can monitor data by reading it and dumping the data to a JSON file. Details are in the upcoming sections.

## Dependencies:

## modbus-tk:

The Tristar MPPT uses a royalty-free serial protocol called MODBUS. There exist many libraries to read it. The Python language has pymodbus, and modbus-tk. Since pymodbus is not reliable (and I want reliable code), I will be using the modbus-tk library. It is distributed under the GNU-LGPL license (GNU Lesser General Public License) © 2009. Created by Luc Jean – [luc.jean@gmail.com](mailto:luc.jean@gmail.com) and Apidev – [http://www.apidev.fr](http://www.apidev.fr/). No warranty of any kind.

### json:

Comes with every distribution of python. Necessary to convert dictionaries into JSON format and dump it directly to an outfile.

### serial:

This is a serial library for Python. It’s easy to use, and free. Provided as-is. Install with pip install pyserial. Use by calling “import serial”. ©2015 Chris Liechi [clichi@gmx.net](mailto:clichi@gmx.net) All Rights Reserved.

# Initialization

The Morningstar() class (like most other classes) has an \_\_init\_\_() function, that calls itself whenever a Morningstar() object is created. It requires port, baudrate, and the MODBUS slave number as arguments. When a Morningstar() object is created, it will initialize the serial connection to the PLC using this information. Then, it will create a MODBUS RtuMaster() class from the MODBUS-TK library.

After the MODBUS RtuMaster() class is initialized, it will call the internal function .scaling(). It will test what scaling factors are used.

Dictionaries in Python are everywhere in this code.

It will have methods to either dump data to the command line or dump data to an outfile. The outfile should be a .json file, since the contents will be written in JSON format.

# Classes:

## Morningstar()

### Description:

Reads data from the Tristar MPPT PLC. Takes PORT, BAUDRATE, and SLAVE\_NUMBER upon initialization.

### Variables:

PORT: what port number are you using?

BAUD: what baudrate are communicating at?

SLAVE\_NUMBER: what is the MODBUS slave number you’re reading from?

serial\_connection: the serial connection from pyserial that actually communicates with the PLC.

master: the class from modbus\_tk that converts serial data into data we can read.

### Methods:

**.scaling()**: Sets the classes internal scaling properties (V\_PU and I\_PU). Also prints the current running version to the console. Runs every time an object of this class is created.

**.ADCdata()**: Returns a dictionary containing ‘battery voltage’, ‘battery terminal voltage’, ‘battery sense voltage’, ‘array voltage’, ‘battery current’, ‘array current’, ‘12V supply’, ‘3V supply’, ‘meterbus voltrage’, ‘1.8V supply’, and ‘reference voltage’.

**.TemperatureData()**: Returns a dictionary containing ‘heatsink temperature’, ‘RTS temperature’, and ‘battery regulation temperature’. All are in degrees Celsius.

**.StatusData()**: Returns a dictionary containing ‘battery\_voltage’, ‘charging\_current’, ‘minimum battery voltage’, ‘maximum battery voltage’, ‘hour meter’, a list of faults, a list of alarms, the current state of the DIP switch, and the current state of the LED.

**.ChargerData()**: Returns a dictionary containing ‘Charge State’, ‘target regulation voltage’, ‘Ah Charge Resettable’, ‘Ah Charge Total’, ‘kWhr Charge Resettable’, and ‘kWhr Charge Total’.

**.MPPTData()**: Returns a dictionary containing: ‘output power’, ‘input power’, ‘max power of last sweep’, ‘Vmp of last sweep’, and ‘Voc of last sweep’.

**.Logger\_TodaysValues()**: Returns a dictionary containing: ‘Battery Voltage Minimum Daily’, ‘Battery Voltage Maximum Daily’, ‘Input Voltage Maximum Daily’, ‘Amp Hours Accumulated Daily’, ‘Watt hours accumulated daily’, ‘Maximum power output daily’, ‘Minimum temperature daily’, ‘Maximum Temperature Daily’, a list of daily faults, a list of daily alarms, ‘time\_ab\_daily’, ‘time\_eq\_daily’, and ‘time\_fl\_daily’.

**.ChargeSettings()**: Returns a dictionary containing: 'EV\_absorp', 'EV\_float', 'Et\_absorp', ‘Et\_absorp\_ext', 'EV\_absorp\_ext', 'EV\_float\_cancel', 'Et\_float\_exit\_cum', 'EV\_eq'], 'Et\_eqcalendar', 'Et\_eq\_above', 'Et\_eq\_reg', 'Et\_battery\_service', 'EV\_tempcomp', 'EV\_hvd', 'EV\_hvr', 'Evb\_ref\_lim', 'ETb\_max', 'ETb\_min', 'EV\_soc\_g\_gy', 'EV\_soc\_gy\_y', 'EV\_soc\_y\_yr', 'EV\_soc\_yr\_r', 'Elb\_lim', 'EVa\_ref\_fixed\_init', 'Eva\_ref\_fixed\_pet\_init'

**.DumpInstantenousDataToJSONFile(outfile)**: Calls all instantaneous data internal class methods (ADCdata(), TemperatureData(), StatusData(), ChargerData(), MPPTData()), and dumps them into an outfile using json.dumps(). Preferably, the file’s name will end in “.json” so the operating system can recognize that the file is in JSON format.

**.DumpDailyDataToJSONFile(outfile)**: Calls all daily data internal class methods (Logger\_TodaysValues() and ChargeSettings()) and dumps them into an outfile using json.dumps(). Preferably, the file’s name will end in “.json” so the operating system can recognize that the file is in JSON format.

BMS Monitoring Code Design Document

# BMS.py Design Document

Author: William Kerr

# Introduction:

The Battery Management System (BMS) has a Programmable Logic Controller (PLC) in it. It can be used to monitor things such as charge dissipated, voltage levels of each individual cell, etc.

The internal PLC can be monitored using a USB interface. This is what the Raspberry Pi will do using the interpreted Python Language.

The Python language is dependent on classes to process data. So, I will be writing a Python class to extract data from the BMS PLC.

## Dependencies:

### pyserial:

This is a serial library for Python. It’s easy to use, and free. Provided as-is. Install with pip install pyserial. Use by calling “import serial”. ©2015 Chris Liechi [clichi@gmx.net](mailto:clichi@gmx.net) All Rights Reserved.

Usage: <https://pyserial.readthedocs.io/en/latest/pyserial.html>

### crc8\_dallas:

This is a CRC-8 library that uses the exact polynomial we need for this application: x^8 + x^2+x+1. I had to modify the code to work with Python 3, since it was originally developed for Python 2.

### sys:

Comes with every distribution of python. Necessary to have a test bench.

Usage: <https://docs.python.org/3/library/sys.html>

### json:

Comes with every distribution of python. Necessary to convert dictionaries into JSON format and dump it directly to an outfile.

Usage: <https://docs.python.org/3/library/json.html>

# Initialization:

Create a BMS() object, passing in PORT and BAUDRATE. This will initialize the serial connection to the BMS PLC.

The BMS() object will destroy itself when python exits.

# Classes:

## BMSStatistic():

### Description:

An internal class that contains a statistic from the sentence SS1(). Makes it easier to do mass data collection from a series of sentences if a request for every statistic available is made.

### Variables:

Every BMSStatistic object contains at least 4 variables:

* statisticIdentifier: what is the ID of this statistic (i.e. what protocol to use to process it)
* statisticValue: what is the value spat out (in decimal converted earlier from hexadecimal) from the BMS system?
* statisticValueAdditionalInfo: any additional information spat out from the BMS system (e.g. Cell ID)?
* timestamp: what time (in seconds since January 1, 1970 at 00:00 GMT) recorded. The BMS system records it in seconds since January 1, 2000 at 00:00 GMT).

Possible additional variables the class can have:

* Name: What is the real name of the statistic?
* Unit: what unit is the value recorded in (e.g. V, mA, W)? If N/A, the value is simply how many times an event occurred.
* Cell\_ID: What is the ID of the cell the statistic came from?

### Methods:

* .dict(): converts this class into a dictionary with keys being the class variables it has, and their corresponding values.
* .string(): converts this class into a string in JSON format.
* .\_\_init\_\_(): initializes the object. Takes statisticIdentifier,statisticValue,statisticValueAdditionalInfo,timestamp. Upon creation, runs a specific protocol to process the data based on its statisticIdentifier.

## BMS():

### Description:

A class that can read the BMS system. Call the .DumpToJSONFile() method to dump all data to an outfile. Details below.

### Variables:

Every BMS() object contains at least 3 variables:

* PORT: what port number is the Raspberry Pi reading from?
* BAUDRATE: at what baudrate (in bits/second) is the Raspberry Pi reading at?
* ser: the serial object (from the pyserial library) that sends and receives data from the BMS.

### Methods:

* .VR1(): returns a dictionary containing hardware type, serial number, and firmware version.
* .BB1(): returns a dictionary containing number of cells, minimum balancing rate, and average cell balancing rate.
* .BB2(): returns a dictionary containing cell string number, first cell number, size of group, and individual cell module balancing rate of each cell group.
* .BC1(): returns a dictionary containing battery charge, battery capacity, and state of charge.
* .BT1(): returns a dictionary containing the summary of cell module temperature values of the battery pack.
* .BT2(): This sentence contains individual cell module temperatures of a group of cells. Each group consists of 1 to 8 cells. This sentence is sent only after Control Unit receives a request sentence from external device, where the only data field is ‘?’ symbol. The normal response to BT2 request message, when battery pack is made up of two parallel cell strings:
* .BT3(): This sentence contains the summary of cell temperature values of the battery pack. It is sent periodically with configurable time intervals for active and sleep states (Data Transmission to Display Period).
* .BT4(): This sentence contains individual cell temperatures of a group of cells. Each group consists of 1 to 8 cells.
* .BV1(): Returns a dictionary containing a summary of cell voltages. contains number of cells, minimum cell voltage, maximum cell voltage, average cell voltage, and total voltage.
* .BV2(): This sentence contains individual voltages of a group of cells. Each group consists of 1 to 8 cells.
* .CF2(parameterID): returns the parameter data of the parameter ID. Must be processed separately.
* .CG1(): This sentence contains the statuses of Emus internal CAN peripherals. Can include CAN current sensor, and CAN cell group, along with the cell group number.
* .CN1(): This sentence reports the CAN messages received on CAN bus by Emus BMS Control Unit, if “Send to RS232/USB” function is enabled.
* .CN2(): This sentence reports the CAN messages sent on CAN bus if "Send to RS232/USB function is enabled.
* .CS1(): Returns a dictionary containing the parameters and status of the charger. Includes set voltage, set current, actual voltage, actual current, number of connected charger, and CAN charger status.
* .CV1(): Returns a dictionary containing the values of total voltage of battery pack, and current flowing through the battery pack.
* .DT1(): This is a placeholder for an electric vehicle sentence. The code is being specifically programmed for a greenhouse, so this sentence will not be programmed and return an error.
* .FD1(): This sentence resets the unit to factory defaults. Use at your own risk.
* .IN1(): This sentence returns a dictionary containing the status of the input pins (AC sense, IGN In, FAST\_CHG).
* .LG1(clear): This sentence can either: retrieve events logged, or clear the event logger.
  + Retrieve Events Logged: pass in ‘N’ or a null value.
    - Every event is recorded in a dictionary form like this: [“log event number 1”]: [“log event”: “No event”, “unix time stamp”: 1567014467
    - Possible events:
      * No Event
      * BMS started
      * Lost communication to cells
      * Established communication to cells
      * Cells voltage critically low
      * Critical low voltage recovered
      * Cells voltage critically high
      * Critical high voltage recovered
      * Discharge current critically high
      * Discharge critical high current recovered
      * Charge current critically high
      * Charge critical high current recovered
      * Cell module temperature critically high
      * Critical high cell module temperature recovered
      * Leakage detected
      * Leakage recovered
      * Warning: low voltage – reducing power
      * Power reduction due to low voltage recovered
      * Warning: high current – reducing power
      * Power reduction due to high current recovered
      * Warning: High Cell module temperature – reducing power
      * Power reduction due to high cell module temperature recovered.
      * Charger connected
      * Charger disconnected
      * Started pre-heating stage
      * Started pre-charging stage
      * Started main charging stage
      * Started balancing stage
      * Charging finished
      * Charging error occurred
      * Retrying charging
      * Restarting charging
      * Cell Temperature Critically high
      * Critically high cell temperature recovered
      * Warning: High cell temperature – reducing power
    - Unix Timestamp: Time recorded in seconds since January 1, 1970 at 00:00 GMT.
    - Log event number: what event number it
  + Clear Event Logger: pass in the ascii value ‘C’ or ‘c’.
* .OT1(): Returns a dictionary containing the status of output pins (Charger pin, heater, bat. low, buzzer, chg. ind.)
* .PW1(request, password): Check the admin status with PW1(‘?’). Log into BMS system with PW1(‘P’, password). Logout with PW1().
* .PW2(request, newPassword): Sets a new password, or clears a password. To set new password, call PW2('S',"mynewpassword"), and substitute “mynewpassword” with whatever password you want. To clear your password, call PW2('C'). Returns true if successful, false if not successful.
* .RC1(): Resets the current sensor reading to zero. Used after current sensor is initially installed.
* .RS1(): Resets the Emus BMS control unit entirely. Like a sudo reboot on a linux machine. Requires admin clearance.
* .RS2(): This sentence is used to retrieve the reset source history log.
* .SC1(percentage): This sentence sets the current state of the charge of the battery in %. Send in an integer from 0 to 100. This method will convert to hexadecimal format first. Returns False if not successful or invalid percentage is passed. Returns True if successful.
* .SS1(request, statisticIdentifier): This sentence can either: Request All Statistics, Request a Specific Statistic (pass in a number), or Clear all unprotected statistics.
  + Request All Statistics: call SS1(‘?’). This will return all statistics the BMS currently has in the form of dictionaries converted from BMSstatistic classes.
  + Request a Specific Statistic: call SS1(‘N’, number), where number is a positive integer. Returns a dictionary containing a single statistic.
  + Clear all unprotected statistics: call SS1(‘c’).
* .ST1(): This sentence returns the status of the BMS in dictionary form. It contains these statistics:
  + **Charging flags**: charging stage, last charging error, last charging error parameter (for debugging purposes), stage duration,
  + **Status flags:** Valid cell voltages, Valid balancing rates, valid number of live cells, battery charging finished, valid cell temperatures
  + **Protection flags**: undervoltage, overvoltage, discharge overcurrent, charge overcurrent, cell module overheat, leakage, no\_cell\_comm, cell\_overheat
  + **Power flags**: warning: power reduction: low voltage, warning: power reduction: high current, warning: power reduction: high cell module temperature, warning: power reduction: high cell temperature
  + **Pin flags**: no\_function, speed\_sensor, fast\_charge\_switch, ign\_key, charger\_mains\_AC\_sense, heater\_enable, sound\_buzzer, battery\_low, charging\_indication, charger\_enable\_output, state\_of\_charge, battery\_contactor, battery\_fan, current\_sensor, leakage\_sensor, power\_reduction, charging\_interlock, analog\_charger\_control, ZVU\_boost\_charge, ZVU\_slow\_charge, ZVU\_buffer\_mode, BMS\_failure, equalization\_enable, DCDC\_control, ESM\_rectifier\_current\_limit, contactor\_precharge
* .TD1(): Returns time and date according the BMS in dictionary form. Returns year, month, day, hour, minute, second, and the amount of uptime the unit has in seconds.
* .TC2(): Used to calibrate cell temperature by a PC, not a microcontroller.
* .DumpToJSONfile(outfile): Calls all data methods listed above, then dumps all data returned into an outfile in JSON format.

Note: Every data harvesting method returns “Cannot communicate to cells” if it fails.

# Private Methods:

bitAt(bitfield, position):

### Description:

Returns True if the bit is 1 at the position of the bitfield, False of 0. Used to analyze bitfields with fewer lines.

Schematic:

A close up of a map

Description automatically generatedA close up of a map

Description automatically generatedA close up of a map

Description automatically generatedA screenshot of a cell phone

Description automatically generatedA close up of a map

Description automatically generated