Morningstar.py Design Document

Author: William Kerr

Table of Contents

[Introduction: 2](#_Toc17885271)

[Dependencies: 2](#_Toc17885272)

[modbus-tk: 2](#_Toc17885273)

[json: 2](#_Toc17885274)

[serial: 2](#_Toc17885275)

[Initialization 2](#_Toc17885276)

[Classes: 3](#_Toc17885277)

[Morningstar() 3](#_Toc17885278)

[Description: 3](#_Toc17885279)

[Variables: 3](#_Toc17885280)

[Methods: 3](#_Toc17885281)

# 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.