**Raspberry Pi Python Code Repository:** [**https://github.com/BUSolarLab/EDSFieldController**](https://github.com/BUSolarLab/EDSFieldController)

**PYTHON CODE DOCUMENTATION**

The python code is split into the following files:

MasterManager.py

TestingManager.py

DataManager.py

StaticManager.py

SP420.py

AM2315.py

The following documentation describes, in detail, the functionality of each of the above files. All other files used by the python code are external dependencies that will be outlined at the end.

***MasterManager.py-----------------------------------------------------------------------------------------------------***

This is the central loop file that controls the overall structure of the field test unit functionality. It consists of a series of initialization checks, class instantiations, and variable setups, followed by a main looping structure that runs all routine functions.

*-Code Outline-*

Initialization

Loop:

Repeating Initialization

Check Time

Flip Green LED Indicator

Check for Solar Noon

If Yes: Run Solar Voc/Isc

Check for Scheduled Tests

If Yes: Run EDS Tests

Check for Manual Activation

If Yes: Run EDS Test on Manual

Write Error to Log (if any)

*-Pseudocode-*

**Execute code**

**Initialization()**

The code checks for an existing configuration file from which to load predetermined parameters that control various user-controlled aspects of the EDS, from testing schedules to Raspberry Pi pinout settings. If the file does not exist, or if there is an error reading one that does exist, it will create a new configuration file using hard-coded default parameters. The code will also check that a USB has been inserted to the Raspberry Pi for data collection purposes, and will throw an error if not. It is recommended that the initial code execution be done with the Raspberry Pi connected to a screen, so the user can monitor the code output for the purpose of error checking and feedback.

**Core Loop: (until STOP)**

**Check components exist()**

The code will first make a check to ensure that each component necessary for the field unit functionality exists and is working properly. This includes:

* the Real Time Clock (RTC)
* the weather sensor
* ADC input from a PV cell.

If any component is not responding within prescribed limits, an error message will be displayed the error indicator light (red LED) will be set to blink to alert personnel to check the system manually. The code will also be halted at this point in the case of an error.

**Repeating Initialization()**

The code will reset all relays to make sure they are switched off. This is a measure against accidental relay activation, since the standard operation does not have relays activated until tests are made.

**IF no error found:**

**Blink green LED()**

The all-clear green LED will blink if the system is working properly. This blink will match the frequency of the core loop execution, and will be approximately 1 sec ON and 1 sec OFF in sequence.

**End IF**

**Check time of day()**

The code will communicate with the Real Time Clock (RTC) onboard the Raspberry Pi to get the current time and date. Because the unit will not always be in range of a WiFi signal, and it requires daily timing for scheduling EDS activation tests, we have included an RTC to maintain proper time and date independently of an active internet connection.

**IF time matches schedule:**

**Check weather()**

The code will then check for proper weather conditions as outlined in the configuration file by communicating with the attached weather sensor. If the temperature and humidity values are within the acceptable range, a flag is set to allow for EDS testing to continue.

**Main testing loop: (for each EDS being tested)**

**Solid green LED()**

During a test, the green indicator LED will be lit and NOT blink, to distinguish from regular code function.

If both time and weather checks are good, the code will execute the central testing loop, which consists of a series of steps necessary to control the electronics to activate the EDS and receive data. The list of steps is as follows:

1) Pull open circuit voltage and short-circuit current data from ADC for each EDS being tested, as well as for the control PV cell.

2) Execute low-power switching and close each relay for each EDS to connect the power supply circuit output and activate the EDS.

3) After testing time has elapsed (2 min standard), low-power switch each relay open to stop the test.

4) Pull short-circuit current data again

The code will also record the data in a .csv file that is updated for each test carried out. If an error is encountered at any time during the test, the test will be stopped and the data thrown out.

**End IF**

**Check if manual testing switch is activated()**

There is a switch to manually power an EDS in order to activate on the user's command, if wanted. The code will check that this switch has been flipped, and then execute the main testing loop (outlined above) if so.

The core loop does not have an internal function to STOP unless an error is encountered or until a maximum time limit is achieved (set in the configuration file). The user can cease code function by powering off the Raspberry Pi, or by stopping the code manually from within the Raspberry Pi.

**End Code**

***-------------------------------------------------------------------------------------------------------------------------------***

***TestingManager.py----------------------------------------------------------------------------------------------------***

This peripheral file contains code classes dedicated to all testing functionality of the field test unit. In particular, this code coordinates all relay testing procedures, as well as checks for schedule times and weather for initiating tests. It also contains measurement procedures for PV cell Voc/Isc, and communication with the ADC circuit.

*-Code Outline-*

ADC Class: ADCMaster

Testing Procedure Class: TestingMaster

*-Pseudocode-*

**ADCMaster Class:**

**get\_ocv\_PV()**

Reads input to the ADC, which represents the open circuit voltage from the PV cell being tested. This will be determined by the currently selected transistor within the ADC circuit board.

**get\_scc\_PV()**

Reads input to the ADC, which now represents the short circuit current from the PV cell being tested. This is determined by an alternately selected transistor from the ADC circuit board.

**get\_ocv\_BAT()**

Reads the input to the ADC, which is the battery’s voltage reading. This is on a different ADC channel, and is always available.

**TestingMaster Class:**

**check\_time()**

Checks the current time against each of the schedules for each EDS. If the current time is within a small window of the scheduled time, it will pass the check (this window can be set by changing MIN\_CHECK\_THRESHOLD, and is currently 30 seconds).

**check\_temp() / check\_humid()**

Checks the weather (temp and humidity) against the currently set range for acceptable testingconditions. If these conditions are not met, it will loop for a set duration, checking for the proper conditions until they are met. This is to prevent skipping a test due to not being exactly within the prescribed range of temperature and humidity.

**run\_test() / run\_test\_begin() / run\_test\_end() / run\_measure\_\*()**

These methods coordinate the relay switching that must occur for each testing sequence. The standard testing procedure is carried out with the following steps:

1: Delay

2: Flip relay ON for Control PV (setup GPIO channel as OUTPUT)

3: Delay

4: Drain OCV transistor branch on ADC circuit (send HIGH logic)

5: Read ADC value on main Channel [pre-test OCV]

6: Reset transistor branch (send LOW logic)

7: Delay

8: Drain SCC transistor branch on ADC circuit (send HIGH logic)

9: Read ADC value on main Channel [pre-test SCC]

10: Reset SCC transistor branch (send LOW logic)

11: Delay

12: Flip relay OFF for Control PV (cleanup GPIO channel)

13: Delay

*Repeat 1 - 13 for each control PV being used (default is CTRL1, CTRL2)*

*Repeat 1 - 13 for EDS being tested [pre-test values]*

14: Delay

15: Flip phase relays ON for EDS (setup GPIO channel as OUTPUT)

16: Delay

17: Flip PS relay ON (send HIGH logic)

18: Wait for full testing time (default is 2 minutes)

19: Flip PS relay OFF (send LOW logic)

20: Delay

21: Flip phase relays OFF for EDS (cleanup GPIO channel)

22: Delay

*Repeat 1 - 13 for EDS being tested [post-test values]*

*Repeat 1 - 13 for each control PV being used [post-test values]*

23: Post and record all printed data

***-------------------------------------------------------------------------------------------------------------------------------***

***DataManager.py--------------------------------------------------------------------------------------------------------***

This file contains all code for communicating with an inserted USB drive, which includes writing .txt and .csv data as well as a log file for all activity. It requires that the usbmount dependency is downloaded and installed on the current Raspbian OS with the necessary configuration file changes. The DataManager file also includes a calculation for minutes offset from solar noon, allowing proper sun position normalization for testing across the globe.

*- Code Outline -*

USB Communication Class: USBMaster

Data Writing Class: CSVMaster

Log File Writing Class: LogMaster

Solar Time Offset Calculation Function

*- Pseudocode -*

**USBMaster Class:**

**set\_usb\_name()**

Automatically sets up a path for all files being written to the currently inserted USB drive. This class contains some vestigial code from when a different method of communication was being tested with the USB. The class constructor only executes the necessary methods for the current scheme to function.

**CSVMaster Class:**

**check\_for\_\*\_file()**

These functions check to see if there exists a file that has already been made of each type, so that data is appended to those files instead of new files being made every time a new set of test data is taken. If any files don’t yet exist (a new USB inserted, old files deleted, etc) this function will create empty files with the headers defined in the variables section of DataManager.

**write\_\*\*\*\_data()**

These functions take input datastreams and write them to their respective data files on the USB. An additional consideration might be made for also writing all the data onto the RPi onboard memory for the sake of redundancy, but that is not currently implemented.

**LogMaster Class:**

**check\_for\_log\_file()**

Same functionality as for CSVMaster.

**write\_log()**

Takes a given text stream and writes it to the next line in the log file.

**get\_solar\_time()**

This function takes current time and geopositional data (config file must be changed to reflect the current testing location) and performs a calculation to find the time (in minutes) offset from solar noon for that location. The local time will be adjusted by this value to normalize testing schedules to the same sun position each day.

***-------------------------------------------------------------------------------------------------------------------------------***

***StaticManager.py-------------------------------------------------------------------------------------------------------***

This file is used for reading and loading variables from the JSON configuration file. Values from this file control many aspects of the automated and testing procedure, and should be checked thoroughly before the unit is sent off for extended testing. A description of each configuration variable is as follows:

**SCHEDS1, SCHEDS2, SCHEDS3, SCHEDS4, SCHEDS5, SCHEDS6**, (extra as SCHEDS#)

*Sets the schedule for each EDS being tested. The format for the testing schedule is:*

*[ number of days between tests plus 1 , number of hours offset from solar noon for that test ]*

*Multiple tests can be scheduled by using a nested format: [ [ test 1 , test 1 ] , [ test 2 , test 2 ] ]*

**EDS1, EDS2, EDS3, EDS4, EDS5, EDS6**, (extra as EDS#)

*Sets the GPIO pin output number for controlling the relays for the triple-phase power to each EDS being tested. Only one relay is needed per triplet, so only a single pin is used per EDS.*

**EDS1PV, EDS2PV, EDS3PV, EDS4PV, EDS5PV**, (extra as EDS#PV)

**CTRL1PV, CTRL2PV**, (extra as CTRL#PV)

*Sets the GPIO pin output number for controlling relays connected to PV cells under each numbered EDS or Control panel. These are for selecting the proper PV to be measured by the ADC circuit during data collection.*

**EDSIDS, CTRLIDS**

*These lists set the EDS or Controls actually used during the autonomous testing period. The schedules for unlisted EDS will not be checked, and so make sure all the EDS that need testing are included in the list (unordered is okay).*

**POWER**

*GPIO pin output number for Power Supply relay that controls if 12V is given to the Power Supply for conversion into three phase power output to EDS.*

**OCVBRANCH, SCCBRANCH**

*Set the GPIO pin output numbers for the corresponding transistor branches in the ADC circuit. By sending HIGH logic to one of these branches its transistor will drain and either open circuit voltage (OCV) or short circuit current (SCC) will be measured by the ADC chip.*

**max/minTemperatureCelsius, max/minRelativeHumidity**

*Set the acceptable testing range for temperature and humidity. Tests will only execute if current weather conditions fall within both ranges (with a slight fudge factor that is determined in the TestingManager file).*

**ADCResMain, ADCResOCV, ADCResSCC, ADCBatteryDiv**

*The first three parameters are resistor values used in the ADC circuit. Main is the central resistor that is connected directed to the ADC chip channel. The OCV and SCC values are resistances of each of their respective branches. The SCC branch resistor is as low as possible for the purposes of approximating a short circuit current reading. The BatteryDiv value is the quotient for the voltage divider circuit from the direct battery measurement, and is set as the voltage divider ratio instead of separate resistances because resistor values for this divider are very high (megaOhms).*

**inPinManualActivate, manualEDSNumber**

*Activate sets the GPIO pin input number for measuring if the switch has been flipped for a manual activation event. Number sets the EDS number that will be activated when the manual switch is flipped. This number is ONLY A SINGLE NUMBER (NO LIST), since multiple simultaneous activations have not yet been vetted for effectiveness.*

**degLongitude, offsetGMT**

*These values are used to determine the solar noon offset and are dependent on the current testing location. These are location specific and must be set properly or the scheduled tests will not be coordinated for normalized sun position.*

*- Code Outline -*

Setting up default configuration file parameters (in case file is missing and needs to be created)

Configuration File Management Class: StaticMaster

**StaticMaster Class:**

**load\_config()**

If a configuration file exists, it will read the file’s contents and load them into the current program memory. If the file does not exist, it will create a default configuration file using the ascribed default values

**check\_parameters()**

After loading the configuration file data, this method ensures that each parameter named in the default parameters list exists within the loaded data from the external config file. It does not check that the values are the same, because the values can be edited by simply editing the configuration file itself, and so values won’t necessarily need to match the default values given in the StaticManager code. If any named parameter does not exist within the data loaded from the external config, the external config file will be appended with the missing parameter at its default value and the whole config file will be loaded again.