Artifact Information

Artifact ID	Artifact Title
SRC-001	Software Source Code

Capstone Team	Revision	Artifact Date
Capstone Team 27 - Granustem	1.1	Apr 13, 2019

Prepared by	Checked by
Jonathan Meldrum	Tanner Gaskin

Revision History

Revision #	Date	Prepared by	Checked by	Description	Approved by
1.0	Mar 01, 2019	Jonathan Meldrum	Tanner Gaskin	Software files as of Mar 1, 2019 from the team's GitHub repository	Reese Bastian
1.1	Apr 13, 2019	Jonathan Meldrum	Tanner Gaskin	Software files as of Apr 13, 2019 from the team's GitHub Repository	Reese Bastian

```python
from connections import *

def accl_test():
    print("\n **********Beginning ACCL Test********** \n")
    sleep_time = .1
    for i in range(int(3.0/sleep_time)):
        x, y, z = [value / adafruit_lis3dh.STANDARD_GRAVITY for value in \
                    lis3dh.acceleration]
        print("x = %0.3f G, y = %0.3f G, z = %0.3f G" % (x, y, z))

        sleep(sleep_time)

    print("\n **********Ending ACCL Test********** \n")

if __name__ == "__main__":
    accl_test()

```

```python
from connections import *

def adc_test():
    print("\n **********Beginning ADC Test********** \n")
    for i in range(4):
        print("CHAN0 Sample ", i, " value: ", CHAN0.value, " voltage:", CHAN0.voltage)
        print("CHAN1 Sample ", i, " value: ", CHAN1.value, " voltage:", CHAN1.voltage)
        print("CHAN2 Sample ", i, " value: ", CHAN2.value, " voltage:", CHAN2.voltage)
        print("CHAN3 Sample ", i, " value: ", CHAN3.value, " voltage:", CHAN3.voltage)
        print("CHAN4 Sample ", i, " value: ", CHAN4.value, " voltage:", CHAN4.voltage)
        print("CHAN5 Sample ", i, " value: ", CHAN5.value, " voltage:", CHAN5.voltage)
        print("CHAN6 Sample ", i, " value: ", CHAN6.value, " voltage:", CHAN6.voltage)
        print("CHAN7 Sample ", i, " value: ", CHAN7.value, " voltage:", CHAN7.voltage)
        print()
        sleep(.5)
    print("\n **********Ending ADC Test********** \n")


if __name__ == "__main__":
    adc_test()

```

```python
from kivy.uix.screenmanager import Screen

class BaseScreen(Screen):
    '''Keeps track of the screen history to allow users to move to the previous screen,
    rather than having to specify which screen to move to each time.'''

    screen_history = []

    def move_to(self, screen_name):
        '''Add the current screen to the screen history and move to a new screen.  If we
        are moving to the previous screen, remove it from the screen history.'''
        if self.screen_history and self.screen_history[-1] == screen_name:
            # Make sure to pop the stack if we're moving back to the previous screen
            self.back()
        else:
            self.screen_history.append(self.name)
            self.manager.current = screen_name

    def back(self):
        '''Go to the previous screen.'''
        self.manager.current = self.screen_history.pop()

```

```
<BreakHeightScreen>:
    name: 'break_height_screen'
    GranuContainer:
        GranuSideArea:
            GranuSideButton:
                text: 'Save'
                on_release: if root.save(): root.back()
            GranuSideButton:
                text: 'Cancel'
                on_release:
                    root.back()
            GranuNone:
            GranuNone:
        GranuContent:
            GranuTitle:
                text: 'Break Height'
            GridLayout:
                cols: 2
                rows: 1
                FloatInput:
                    id: break_height
                    font_size: 30
                    size_hint_max_y: 30+15
                Label:
                    text: "cm"
                    size_hint_x: 0.2
                    font_size: 30

```

```python
"""
An input text box that, when selected, allows the user to type in the Break Height value of
the last test via a touch screen number pad that will pop up. The value in the input text box
when you first visit this view is whatever value for the Height setting is currently stored
in our settings file.
"""

from kivy.lang import Builder

import configurator as config
from view.BaseScreen import BaseScreen
from view.input.FloatInput import FloatInput

Builder.load_file('view/screens/main/testing/BreakHeightScreen.kv')

class BreakHeightScreen(BaseScreen):
    def on_pre_enter(self):
        """Before the Screen loads, read the configuration file to get the current
        height."""
        input = self.ids['break_height']
        input.text = str(config.get('break_height', 0))
        input.validate()

    def on_enter(self):
        """Once the Screen loads, focus the TextInput"""
        input = self.ids['break_height']
        input.focus = True

    def save(self):
        """Save button was pressed: save the new height in the configuration file.
        Returns True if save was successful.  False otherwise."""
        input = self.ids['break_height']
        valid = input.validate()
        if valid:
            config.set('break_height', input.text)
            return True
        else:
            input.focus = True
            return False


```

```
<CalibratingPopup>:
    title: 'Calibrating...'
    title_align: 'center'
    size_hint: (0.9, 0.9)
    Label:
        text: 'Reading ADC Data'

<CalibratePointScreen>:
    name: 'calibrate_point_screen'
    GranuContainer:
        GranuSideArea:
            GranuSideButton:
                text: 'Add'
                on_release:
                    if root.add(): root.back()
            GranuSideButton:
                text: 'Cancel'
                on_release:
                    root.back()
            GranuNone:
            GranuSideButton:
                text: 'Calibrate'
                on_release:
                    root.calibrate()
        GranuContent:
            GranuTitle:
                text: 'Add Calibration Point'
            GridLayout:
                rows: 2
                cols: 2
                Label:
                    text: 'ADC: '
                FloatInput:
                    id: adc
                    font_size: 30
                    multiline: False
                    size_hint_max_y: 30+15
                Label:
                    text: 'Real: '
                FloatInput:
                    id: real
                    font_size: 30
                    multiline: False
                    size_hint_max_y: 30+15

```

```python
from kivy.lang import Builder
from kivy.clock import Clock
from kivy.uix.popup import Popup

import configurator as config
from Sensor import Sensor
from view.BaseScreen import BaseScreen
from view.input.StrInput import StrInput

import numpy

Builder.load_file('view/screens/settings/CalibratePointScreen.kv')

INTERVAL = .003
SECOND_CAP = 1/INTERVAL

class CalibratingPopup(Popup):
    pass

class CalibratePointScreen(BaseScreen):
    def __init__(self, **kwargs):
        super(CalibratePointScreen, self).__init__(**kwargs)
        self.sensors = Sensor()

    def on_pre_enter(self):
        adc_input = self.ids['adc']
        adc_input.text = ''
        real_input = self.ids['real']
        real_input.text = ''

    def add(self):
        adc_input = self.ids['adc']
        real_input = self.ids['real']
        if adc_input.validate() and real_input.validate():
            calib_screen = self.manager.get_screen('calibrate_screen')
            calib_screen.add_point(float(adc_input.text), float(real_input.text))
            return True
        else:
            return False

    def calibrate(self):
        calib_screen = self.manager.get_screen('calibrate_screen')
        sensor = calib_screen.get_sensor()

        self._popup = CalibratingPopup()
        self._popup.open()

        data_list = []
        def update_data(dt):
            data = self.sensors.get_all_data()
            data_list.append(data[sensor])
        event = Clock.schedule_interval(update_data, INTERVAL)
        def calibrate_finish(dt):
            event.cancel()
            adc = numpy.float(numpy.average(data_list))
            self.ids['adc'].text = str(adc)
            self._popup.dismiss()
        Clock.schedule_once(calibrate_finish, 1)
```

```
<PointDisplay>
    canvas.before:
        Color:
            rgba: (.0, 0.9, .1, .3) if self.selected else (0, 0, 0, 1)
        Rectangle:
            pos: self.pos
            size: self.size
    text: '(' + str(root.adc) + ', ' + str(root.real) + ')'

<PointsList>
    viewclass: 'PointDisplay'
    SelectableRecycleBoxLayout:


<PointListTitle@Label>
    size_hint_y: None
    height: self.font_size + 5
    font_size: 15

<CalibrateScreen>:
    name: 'calibrate_screen'
    GranuContainer:
        GranuSideArea:
            GranuSideButton:
                text: 'Save'
                on_release:
                    if root.save(): root.back()
            GranuSideButton:
                text: 'Add Point'
                on_release:
                    root.move_to('calibrate_point_screen')
            GranuNone:
            GranuSideButton:
                text: 'Cancel'
                on_release:
                    root.back()
        GranuContent:
            GranuTitle:
                text: root.sensor_name + ' Calibration'
            GridLayout:
                cols: 2
                rows: 1
                size_hint_max_y: 20+15
                Label:
                    text: "Units:"
                    size_hint_x: 0.2
                    font_size: 20
                StrInput:
                    id: units
                    text: root.units
                    width: 20
                    font_size: 20
                    size_hint_max_y: 20+15
            PointListTitle:
                text: 'Calibration Points'
            PointListTitle:
                text: '(ADC, Real)'
            PointsList:
                id: point_list
                list_data: root.points_list
            Label:
                size_hint_y: None
                height: self.font_size + 5
                font_size: 20
                text: 'real = ' + "{:.3E}".format(root.slope) + '*adc + ' + "{:.3E}".format(root.intercept)
```

```python
from kivy.lang import Builder
from kivy.clock import Clock
from kivy.uix.button import Button
from kivy.uix.label import Label
from kivy.uix.boxlayout import BoxLayout
from kivy.properties import StringProperty, ListProperty, NumericProperty

from view.BaseScreen import BaseScreen
from view.SelectableList import SelectableList, SelectableListBehavior, SelectableRecycleBoxLayout

import numpy

import configurator as config

Builder.load_file('view/screens/settings/CalibrateScreen.kv')

class PointDisplay(SelectableListBehavior, Label):
    adc = NumericProperty()
    real = NumericProperty()

class PointsList(SelectableList):
    def update(self, k, val):
        self.data = [{'adc': x[0], 'real': x[1]} for x in self.list_data]

class CalibrateScreen(BaseScreen):
    sensor_name = StringProperty()
    points_list = ListProperty()
    slope = NumericProperty()
    intercept = NumericProperty()
    units = StringProperty()

    def __init__(self, **kwargs):
        super(CalibrateScreen, self).__init__(**kwargs)
        self.config_data = {}

    def set_sensor(self, name):
        self.sensor_name = name
        self.config_data = config.get('sensors', {})
        if name in self.config_data:
            self.points_list = self.config_data[name]['points_list']
            self.slope = self.config_data[name]['slope']
            self.intercept = self.config_data[name]['intercept']
            self.ids['units'].text = self.config_data[name]['units']
        else:
            self.points_list = []
            self.slope = 1
            self.intercept = 0
            self.ids['units'].text = ''

    def get_sensor(self):
        return self.sensor_name

    def add_point(self, adc, real):
        self.points_list.append((adc, real))
        # Calculate line of best fit using Least Square Method
        adc_points = [x[0] for x in self.points_list]
        real_points = [x[1] for x in self.points_list]
        if len(self.points_list) > 1:
            poly = numpy.polyfit(adc_points, real_points, 1) # Linear regression
            self.slope = numpy.float(poly[0])
            self.intercept = numpy.float(poly[1])
        else:
            self.slope = 1.0
            self.intercept = 0.0

    def save(self):
        self.config_data[self.sensor_name] = {
            'slope': self.slope,
            'intercept': self.intercept,
            'points_list': self.points_list,
            'units': self.ids['units'].text
        }
        config.set('sensors', self.config_data)
        return True
```

```
<ColorsScreen>:
    name: 'colors_screen'
    GranuContainer:
        GranuSideArea:
            GranuSideButton:
                text: 'Back'
                on_release: root.back()
        GranuContent:
            GranuTitle:
                text: 'Colors'
            ColorPicker:
                id: 'picker'

```

```python
from kivy.lang import Builder

from view.BaseScreen import BaseScreen

Builder.load_file('view/screens/settings/ColorsScreen.kv')

class ColorsScreen(BaseScreen):
    pass

```

```python
"""
The configurator module adds an interface to read to and write from the configuration
file ('config.json').  Reading and writing settings to the configuration file allows the
configuration to persist beyond the application lifecycle.
"""

import os
import json

CONFIG_FILE = 'config.json'

data = {}

def load():
    """Loads data from the configuration file, if it exists."""
    global data
    if os.path.isfile(CONFIG_FILE):
        with open(CONFIG_FILE) as f:
            data.update(json.load(f))
    else:
        data = {}

def load_from(filepath):
    '''Loads data from a specified configuration file.  Overwrites CONFIG_FILE'''
    global data
    if os.path.isfile(filepath):
        with open(filepath) as f:
            data.update(json.load(f))
            save()
    else:
        data = {}

def save():
    """Saves data to the configuration file."""
    with open(CONFIG_FILE, 'w') as outfile:
        json.dump(data, outfile, indent=4)

def save_as(filepath):
    '''Saves data to the specified file.'''
    with open(filepath, 'w') as outfile:
        json.dump(data, outfile, indent=4)

def set(key, value):
    """Set a key to value in the configuration JSON file."""
    # Set key to value
    data[key] = value
    save()

def get(key, default):
    """Get a value from the configuration JSON file using a key.  If the value does not
    exist, save the default value into the JSON file and return the default."""
    if key in data:
        return data.get(key)
    else:
        set(key, default)
        return default

if __name__ == "__main__":
    """If the configuration module is run as the main program, test the configuration
    module.  These tests ensure the module returns and saves the default value if a key
    is not defined, returns the saved value if a key is defined, and that the
    configuration file contains the values saved.

    WARNING: This will override the configuration file."""
    assert get('a', 3) == 3
    set('b', 5)
    assert get('b', 1) == 5
    set('a', 9)
    save()
    get('a', 33)
    get('b', 33)
    load()
    assert get('a', 2) == 9
    get('c', 21) # The file should now contain 'c': 21
    print("Check that the configuration file contains the key-value pair 'c': 21")

```

```python
import time
import board
import busio
import digitalio
from time import sleep
import RPi.GPIO as GPIO
import serial
import adafruit_gps
import adafruit_lis3dh
import adafruit_am2320
import adafruit_ads1x15.ads1015 as ADS
from adafruit_ads1x15.analog_in import AnalogIn

GPIO.setmode(GPIO.BCM)
i2c = busio.I2C(board.SCL, board.SDA, 115200)
uart = serial.Serial("/dev/ttyS0", baudrate=9600, timeout=3000)

# Temperature and Humidity sensor, off of the I2C pins on bottom right of board
am = adafruit_am2320.AM2320(i2c)

# Accelerometer, top middle of board
lis3dh = adafruit_lis3dh.LIS3DH_I2C(i2c)
lis3dh.range = adafruit_lis3dh.RANGE_2_G


# GPIO, right edge of board
GPIO1 = 4 #BOARD 7, BCM 4
GPIO2 = 17 #BOARD 11, BCM 17
GPIO3 = 27 #BOARD 13, BCM 27
GPIO4 = 22 #BOARD 15, BCM 22

# SPI/GPIO, top edge of board
SPI_CE1 = 7 #BOARD 26, BCM 7
SPI_CE0 = 8 #BOARD 24, BCM 8
SPI_SCLK = 11 #BOARD 23, BCM 11
SPI_MISO = 9 #BOARD 21, BCM 9
SPI_MOSI = 10 #BOARD 19, BCM 10

GPIO_PINS = [GPIO1, GPIO2, GPIO3, GPIO4, SPI_CE1, SPI_CE0,\
        SPI_SCLK, SPI_MISO, SPI_MOSI]

for pin in GPIO_PINS:
    GPIO.setup(pin, GPIO.OUT)


# ADC
ads1 = ADS.ADS1015(i2c, address=0x49, data_rate = 3300, mode=0)
ads2 = ADS.ADS1015(i2c, address=0x48, data_rate = 3300, mode=0)
CHAN0 = AnalogIn(ads1, ADS.P0)
CHAN1 = AnalogIn(ads1, ADS.P1)
CHAN2 = AnalogIn(ads1, ADS.P2)
CHAN3 = AnalogIn(ads1, ADS.P3)
CHAN4 = AnalogIn(ads2, ADS.P0, ADS.P1)
# CHAN5 = AnalogIn(ads2, ADS.P1)
CHAN6 = AnalogIn(ads2, ADS.P3)
CHAN7 = AnalogIn(ads2, ADS.P2)

# Channels for the pot and force sensors
POT_CHAN = CHAN3
X_LOAD_CHAN = CHAN4
Y_LOAD_CHAN = CHAN2

# Scaling factor for the force sensor
FORCE_SENSOR_SCALING = 3556.1878

# GPS
gps = adafruit_gps.GPS(uart, debug=False)
gps.send_command(b'PMTK314,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0')
gps.send_command(b'PMTK220,1000')
try:
    gps.update()
except:
    print("GPS may have a problem. Try Rebooting")

# MOTORS
PWMA = 16 #BOARD 36, BCM 16
AIN1 = 25 #BOARD 22, BCM 25
AIN2 = 20 #BOARD 38, BCM 20

PWMB = 21 #BOARD 40, BCM 21
BIN1 = 5 #BOARD 29, BCM 5
BIN2 = 12 #BOARD 32, BCM 12

PWMC = 13 #BOARD 33, BCM 13
CIN1 = 26 #BOARD 37, BCM 26
CIN2 = 19 #BOARD 35, BCM 19

PWMD = 24 #BOARD 18, BCM 24
DIN1 = 18 #BOARD 12, BCM 18
DIN2 = 23 #BOARD 16, BCM 23

MOTORS = ['A', 'B', 'C', 'D']
IN1 = [AIN1, BIN1, CIN1, DIN1]
IN2 = [AIN2, BIN2, CIN2, DIN2]
PWM = [PWMA, PWMB, PWMC, PWMD]

for in1, in2, pwm in zip(IN1, IN2, PWM):
    GPIO.setup(in1, GPIO.OUT)
    GPIO.setup(in2, GPIO.OUT)
    GPIO.setup(pwm, GPIO.OUT)

```

```python
import datetime

class Dataset:

    def __init__(self, timestamp, x_load, y_load, pot_angle, imu_angle, data_rate):
        self.timestamp = timestamp
        self.x_load = x_load
        self.y_load = y_load
        self.pot_angle = pot_angle
        self.imu_angle = imu_angle

```

```
<GranuContainer>
    orientation: 'horizontal'
    padding: 10

<GranuSideArea>
    rows: 4
    spacing: [0, 10]
    row_default_height: self.height/4 - (30/4.)
    row_force_default: True
    size_hint_x: 0.37

<GranuSideButton>
    font_size: 40
    halign: 'center'
    valign: 'middle'

<GranuContent>
    orientation: 'vertical'
    padding: [10, 0, 0, 0]
    spacing: 10

<GranuTitle>
    size_hint_max_y: 50
    font_size: 40


```

```python
from kivy.lang import Builder

from kivy.uix.boxlayout import BoxLayout
from kivy.uix.gridlayout import GridLayout
from kivy.uix.button import Button
from kivy.uix.widget import Widget
from kivy.uix.label import Label

Builder.load_file('view/elements.kv')
class GranuContainer(BoxLayout):
    pass

class GranuSideArea(GridLayout):
    pass

class GranuSideButton(Button):
    pass

class GranuNone(Widget):
    pass

class GranuContent(BoxLayout):
    pass

class GranuTitle(Label):
    pass

```

```
<ExitScreen>
    name: 'exit_screen'
    GranuContainer:
        GranuSideArea:
            GranuSideButton:
                text: 'Cancel'
                on_release: root.move_to('main_screen')
            GranuSideButton:
                text: 'Exit'
                on_release: app.stop()
            GranuSideButton:
                text: 'Restart'
                on_release:
                    app.stop()
                    app.run()
            GranuSideButton:
                text: 'Shutdown'
                on_release: root.move_to('main_screen')
        GranuContent:


```

```python
"""
Four buttons to select from: Back, Exit, Restart, and Shut Down
"""

from kivy.lang import Builder

from view.BaseScreen import BaseScreen

Builder.load_file('view/screens/main/ExitScreen.kv')

class ExitScreen(BaseScreen):
    pass

```

```
from kivy.clock import Clock
from kivy.uix.textinput import TextInput

import view.keyboard_man as km

class FloatInput(TextInput):
    def __init__(self, **kwargs):
        '''Floats do not need multiple lines to input, set multiline property to
        false.'''
        super(FloatInput, self).__init__(**kwargs)
        self.multiline = False

    def validate(self):
        '''Check that the input can be cast as an int.'''
        test = False
        try:
            fl = float(self.text)
            test = True
        except:
            test = False
        if test:
            self.background_color = (1, 1, 1, 1)
        else:
            self.background_color = (1, .7, .7, 1)
        return test

    def on_text_validate(self):
        '''Called when enter is pressed.'''
        self.validate()
        Clock.schedule_once(self.focus_and_select)

    def on_focus(self, instance, value):
        '''When the FloatInput is focused, show a numeric keyboard.'''
        if value:
            km.show_keyboard(self, 'numeric')

    def focus_and_select(self, *args):
        '''Focus the TextInput and select all of its text.'''
        self.focus = True
        self.select_all()

```

```python
from connections import *

def gpio_test():
    print("\n *********Beginning GPIO Test********** \n")

    for i in range(3):
        print("Blink: ", i)
        for pin in GPIO_PINS:
            GPIO.output(pin, GPIO.HIGH)
        sleep(.5)
        for pin in GPIO_PINS:
            GPIO.output(pin, GPIO.LOW)
        sleep(.5)

    print("\n *********Ending GPIO Test********** \n")


if __name__ == "__main__":
    gpio_test()
```

```python
from connections import *

def gps_test():
    print("\n *********Beginning GPS Test********** \n")



    # Initialize the GPS module by changing what data it sends and at what rate.
    # These are NMEA extensions for PMTK_314_SET_NMEA_OUTPUT and
    # PMTK_220_SET_NMEA_UPDATERATE but you can send anything from here to adjust
    # the GPS module behavior:
    #   https://cdn-shop.adafruit.com/datasheets/PMTK_A11.pdf

    # Turn on the basic GGA and RMC info (what you typically want)
    # gps.send_command(b'PMTK314,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0')
    # Turn on just minimum info (RMC only, location):
    #gps.send_command(b'PMTK314,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0')
    # Turn off everything:
    #gps.send_command(b'PMTK314,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0')
    # Tuen on everything (not all of it is parsed!)
    #gps.send_command(b'PMTK314,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0')

    # Set update rate to once a second (1hz) which is what you typically want.
    # gps.send_command(b'PMTK220,1000')
    # Or decrease to once every two seconds by doubling the millisecond value.
    # Be sure to also increase your UART timeout above!
    #gps.send_command(b'PMTK220,2000')
    # You can also speed up the rate, but don't go too fast or else you can lose
    # data during parsing.  This would be twice a second (2hz, 500ms delay):
    #gps.send_command(b'PMTK220,500')

    # Main loop runs forever printing the location, etc. every second.
    last_print = time.monotonic()
    cnt = 0
    while cnt < 4:
        cnt = cnt + 1
        # Make sure to call gps.update() every loop iteration and at least twice
        # as fast as data comes from the GPS unit (usually every second).
        # This returns a bool that's true if it parsed new data (you can ignore it
        # though if you don't care and instead look at the has_fix property).
        gps.update()
        # Every second print out current location details if there's a fix.
        current = time.monotonic()
        if current - last_print >= 1.0:
            last_print = current
            if not gps.has_fix:
                # Try again if we don't have a fix yet.
                print('Waiting for fix...')
                continue
            # We have a fix! (gps.has_fix is true)
            # Print out details about the fix like location, date, etc.
            print('=' * 40)  # Print a separator line.
            print('Fix timestamp: {}/{}/{} {:02}:{:02}:{:02}'.format(
                gps.timestamp_utc.tm_mon,   # Grab parts of the time from the
                gps.timestamp_utc.tm_mday,  # struct_time object that holds
                gps.timestamp_utc.tm_year,  # the fix time.  Note you might
                gps.timestamp_utc.tm_hour,  # not get all data like year, day,
                gps.timestamp_utc.tm_min,   # month!
                gps.timestamp_utc.tm_sec))
            print('Latitude: {0:.6f} degrees'.format(gps.latitude))
            print('Longitude: {0:.6f} degrees'.format(gps.longitude))
            print('Fix quality: {}'.format(gps.fix_quality))
            # Some attributes beyond latitude, longitude and timestamp are optional
            # and might not be present.  Check if they're None before trying to use!
            if gps.satellites is not None:
                print('# satellites: {}'.format(gps.satellites))
            if gps.altitude_m is not None:
                print('Altitude: {} meters'.format(gps.altitude_m))
            if gps.speed_knots is not None:
                print('Speed: {} knots'.format(gps.speed_knots))
            if gps.track_angle_deg is not None:
                print('Track angle: {} degrees'.format(gps.track_angle_deg))
            if gps.horizontal_dilution is not None:
                print('Horizontal dilution: {}'.format(gps.horizontal_dilution))
            if gps.height_geoid is not None:
                print('Height geo ID: {} meters'.format(gps.height_geoid))

            # print('{0:.6f}, {1:.6f}, sample_{2:d}, #FF0000'.format(gps.latitude, gps.longitude, cnt))
            # print('{0:.6f}, {1:.6f}'.format(gps.latitude, gps.longitude))
    print("\n *********Ending GPS Test********** \n")


if __name__ == "__main__":
    gps_test()

```

```python
from math import sin
from kivy.garden.graph import Graph, MeshLinePlot
graph = Graph(xlabel='X', ylabel='Y', x_ticks_minor=5,
x_ticks_major=25, y_ticks_major=1,
y_grid_label=True, x_grid_label=True, padding=5,
x_grid=True, y_grid=True, xmin=-0, xmax=100, ymin=-1, ymax=1)
plot = MeshLinePlot(color=[1, 0, 0, 1])
plot.points = [(x, sin(x / 10.)) for x in range(0, 101)]
graph.add_plot(plot)
```

```
<HeightScreen>:
    name: 'height_screen'
    GranuContainer:
        GranuSideArea:
            GranuSideButton:
                text: 'Save'
                on_release: if root.save(): root.back()
            GranuSideButton:
                text: 'Cancel'
                on_release:
                    root.back()
            GranuNone:
            GranuNone:
        GranuContent:
            GranuTitle:
                text: 'Height'
            GridLayout:
                cols: 2
                rows: 1
                FloatInput:
                    id: height
                    font_size: 30
                    size_hint_max_y: 30+15
                Label:
                    text: "cm"
                    size_hint_x: 0.2
                    font_size: 30
```

```python
"""
An input text box that, when selected, allows the user to type in the current Height
setting via a touch screen number pad that will pop up. The value in the input text box
when you first visit this view is whatever value for the Height setting is currently
stored in our settings file.
"""

from kivy.lang import Builder

import configurator as config
from view.BaseScreen import BaseScreen
from view.input.FloatInput import FloatInput

Builder.load_file('view/screens/settings/HeightScreen.kv')

class HeightScreen(BaseScreen):
    def on_pre_enter(self):
        """Before the Screen loads, read the configuration file to get the current
        height."""
        input = self.ids['height']
        input.text = str(config.get('height', 0))
        input.validate()

    def on_enter(self):
        """Once the Screen loads, focus the TextInput"""
        input = self.ids['height']
        input.focus = True
        input.select_all()

    def save(self):
        """Save button was pressed: save the new height in the configuration file.
        Returns True if save was successful.  False otherwise."""
        input = self.ids['height']
        valid = input.validate()
        if valid:
            config.set('height', float(input.text))
            return True
        else:
            input.focus = True
            return False
```

```python
from .connections import *

class Humidity:

    def __init__(self):
        self.hum = 0.0

    def get_data(self):
        try:
            self.hum = am.relative_humidity
            return self.hum
        except:
            return self.hum

```

```python
from .connections import *

class IMU:

    def __init__(self):
        pass

    def get_data(self):
        x, y, z = [value / adafruit_lis3dh.STANDARD_GRAVITY for value in \
                    lis3dh.acceleration]
        # return "x = {0:0.3f} G \n y = {1:0.3f} G \n z = {2:0.3f} G".format(x, y, z)
        return x

```

```
{
    "title" : "Integer",
    "description" : "An integer-only keypad",
    "cols" : 3,
    "rows": 4,
    "normal_1": [
    ["7", "7", "7", 1],
    ["8", "8", "8", 1],
    ["9", "9", "9", 1]],
    "normal_2": [
    ["4", "4", "4", 1],
    ["5", "5", "5", 1],
    ["6", "6", "6", 1]],
    "normal_3": [
    ["1", "1", "1", 1],
    ["2", "2", "2", 1],
    ["3", "3", "3", 1]],
    "normal_4": [
    ["0", "0", "0", 1],
    [" ", null, "", 1],
    ["\u232b", null, "backspace", 1]],
    "shift_1": [
    ["7", "7", "7", 1],
    ["8", "8", "8", 1],
    ["9", "9", "9", 1]],
    "shift_2": [
    ["4", "4", "4", 1],
    ["5", "5", "5", 1],
    ["6", "6", "6", 1]],
    "shift_3": [
    ["1", "1", "1", 1],
    ["2", "2", "2", 1],
    ["3", "3", "3", 1]],
    "shift_4": [
    ["0", "0", "0", 1],
    [" ", null, "", 1],
    ["\u232b", null, "backspace", 1]]
}
```

```python
from kivy.clock import Clock
from kivy.uix.textinput import TextInput

import view.keyboard_man as km

class IntInput(TextInput):
    def __init__(self, **kwargs):
        '''Ints do not need multiple lines to input, set multiline property to
        false.'''
        super(IntInput, self).__init__(**kwargs)
        self.multiline = False

    def validate(self):
        '''Check that the input can be cast as an int.'''
        test = False
        try:
            fl = int(self.text)
            test = True
        except:
            test = False
        if test:
            self.background_color = (1, 1, 1, 1)
        else:
            self.background_color = (1, .7, .7, 1)
        return test

    def on_text_validate(self):
        '''Called when enter is pressed.'''
        self.validate()
        Clock.schedule_once(self.focus_and_select)

    def on_focus(self, instance, value):
        '''When the IntInput is focused, show a numeric keyboard.'''
        if value:
            km.show_keyboard(self, 'integer')

    def focus_and_select(self, *args):
        '''Focus the TextInput and select all of its text.'''
        self.focus = True
        self.select_all()

```

```python
"""
The keyboard manager adds the ability to change keyboard layouts depending on the current
screen.  For example, the plot number screen uses a "numeric" keyboard (number pad),
while the operator uses a "text" keyboard (qwerty).
"""

from kivy.core.window import Window

keyboard = None

def show_keyboard(caller, layout):
    """Shows a keyboard with the specified layout.

    The folder view/keyboard_layouts contains keyboard layouts used in this software,
    including 'numeric' (number pad) and 'text' (qwerty).  These are custom keyboard
    layouts.  Kivy contains its own keyboard layouts; however, Kivy's layouts are
    designed for multi-touch screens."""
    kb = Window.request_keyboard(_close_keyboard, caller)
    if kb.widget:
        keyboard = kb.widget
        if layout=='numeric':
            keyboard.layout = "view/keyboard_layouts/numeric.json"
            keyboard.margin_hint = [0.05, 0.2, 0.05, 0.2]
        elif layout=='integer':
            keyboard.layout = "view/keyboard_layouts/integer.json"
            keyboard.margin_hint = [0.05, 0.2, 0.05, 0.2]
        elif layout=='text':
            keyboard.layout = "view/keyboard_layouts/text.json"
            keyboard.margin_hint = [0.05, 0.06, 0.05, 0.06]
        else:
            keyboard.layout = layout
            keyboard.margin_hint = [0.05, 0.06, 0.05, 0.06]

        # Using internal members is probably not the best way to do this.  But...
        # Turn off capslock each time a new keyboard is requested
        keyboard.have_capslock = False
        keyboard.active_keys.clear()
        keyboard.refresh_active_keys_layer()
    else:
        keyboard = kb

def _close_keyboard():
    """When the keyboard is closed, clear the keyboard global."""
    global keyboard
    if keyboard:
        keyboard = None
```

```
<LiveFeedInfoBox@Label>:
    font_size: 25
    color: 0,0,0,1
    halign: 'center'
    valign: 'center'
    canvas.before:
        Color:
            rgba: .74,.74,.74,1
        Rectangle:
            pos: self.pos
            size: self.size

<LiveFeedScreen>:
    name: 'live_feed_screen'
    GranuContainer:
        GranuSideArea:
            GranuNone:
            GranuNone:
            GranuSideButton:
                text: root.transition_to_state
                on_release:
                    root.transition()
            GranuSideButton:
                text: 'Back'
                on_release:
                    root.manager.transition.direction = 'right'
                    root.move_to('main_screen') # Move to main screen
        GranuContent:
            GranuTitle:
                text: 'Live Feed'
            GridLayout:
                cols: 3
                spacing: 10
                LiveFeedInfoBox:
                    id: temperature
                    text: root.temperature_label + ':\n' + root.temperature + u'\N{DEGREE SIGN}' + "C"
                LiveFeedInfoBox:
                    id: humidity
                    text: root.humidity_label + ':\n' + root.humidity + "%"
                LiveFeedInfoBox:
                    id: location
                    text: root.location_label + ':\n' + root.location
                LiveFeedInfoBox:
                    id: time
                    text: root.time_label + ':\n' + root.time
                LiveFeedInfoBox:
                    id: x_load
                    text: root.x_load_label + ':\n' + root.x_load
                LiveFeedInfoBox:
                    id: y_load
                    text: root.y_load_label + ':\n' + root.y_load
                LiveFeedInfoBox:
                    id: pot_angle
                    text: root.pot_angle_label + ':\n' + root.pot_angle
                LiveFeedInfoBox:
                    id: imu_angle
                    text: root.imu_angle_label + ':\n' + root.imu_angle + "G"
                LiveFeedInfoBox:
                    id: cpu_time
                    text: root.data_rate_label + ':\n' + root.data_rate + " Hz"
```

```python
"""
Shows all data: Temperature, Humidity, Location, Time, and all Sensor data
"""

from kivy.lang import Builder
from kivy.properties import NumericProperty
from kivy.properties import StringProperty
from kivy.properties import ListProperty
from kivy.clock import Clock
from Sensor import Sensor
import datetime

from view.BaseScreen import BaseScreen
from view.elements import *


Builder.load_file('view/screens/main/LiveFeedScreen.kv')

INTERVAL = .004
SECOND_CAP = 1/INTERVAL

class LiveFeedScreen(BaseScreen):
    sensor = Sensor()

    run_count = 0
    transition_to_state = StringProperty("Pause")
    #self.keys = ListProperty()
    #self.values =
    #sensor_keys =  self.sensor.get_sensor_keys()
    #for key in sensor_keys:
    #    self.keys.append(keys)
    #sensor_data =  self.sensor.get_sensor_data()
    #for i in range(0,len(sensor_data)):



    temperature_label = StringProperty("Temperature")
    humidity_label = StringProperty("Humidity")
    location_label = StringProperty("Location")
    time_label = StringProperty("Time")
    x_load_label = StringProperty("X Load")
    y_load_label = StringProperty("Y Load")
    pot_angle_label = StringProperty("Pot Angle")
    imu_angle_label = StringProperty("IMU Angle")
    data_rate_label = StringProperty("Data Rate")

    temperature = StringProperty("0")
    humidity = StringProperty("0")
    location = StringProperty("0.00, 0.00")
    time = StringProperty("00:00:00 AM")
    x_load = StringProperty("0.00")
    y_load = StringProperty("0.00")
    pot_angle = StringProperty("0")
    imu_angle = StringProperty("0")
    data_rate = StringProperty("0")
    old_time = 0

    def on_pre_enter(self):
        self.event = Clock.schedule_interval(self.update_values, INTERVAL)
        self.transition_to_state = "Pause"

    def update_values(self, obj):

        if self.run_count >= SECOND_CAP:
            self.sensor.get_header_data()
            sensor_data = self.sensor.get_sensor_data()
            self.temperature = str(sensor_data["Temperature"])
            self.humidity = str(sensor_data["Humidity"])
            self.location = str(sensor_data["Location"])
            self.time = datetime.datetime.now().strftime("%H:%M:%S %p")
            self.x_load = str(sensor_data["X Load"])
            self.y_load = str(sensor_data["Y Load"])
            self.pot_angle = str(sensor_data["Pot Angle"])
            self.imu_angle = str(sensor_data["IMU Angle"])
            # Calculate Data Acquisition Rate
            now = datetime.datetime.now()
            new_time = (int(now.strftime("%M")) * 60) + int(now.strftime("%S")) + (int(now.strftime("%f"))/1000000)
            time_dif = new_time - self.old_time
            self.data_rate = str(round(SECOND_CAP/time_dif,2))
            self.old_time = new_time
            # Reset run_count
            self.run_count = 0
        else:
            sensor_data = self.sensor.get_sensor_data()
            self.run_count = self.run_count + 1

    def on_leave(self):
        self.event.cancel()

    def transition(self):
        if(self.transition_to_state == "Pause"):
            self.event.cancel()
            self.transition_to_state = "Resume"
        else:
            self.event = Clock.schedule_interval(self.update_values, INTERVAL)
            self.transition_to_state = "Pause"
```

```python
from .connections import *

class Location:

    def __init__(self):
        self.lat = 0.0
        self.long = 0.0
        gps.update()
        if gps.has_fix:
            self.lat = gps.latitude
            self.long = gps.longitude

    def get_data(self):
        gps.update()
        if gps.has_fix:
            self.lat = gps.latitude
            self.long = gps.longitude
        return self.lat, self.long

```

```
# Include custom Kivy widgets for all screens
#:import * view.elements

#:import ExitScreen view.screens.main.ExitScreen
#:import LiveFeedScreen view.screens.main.LiveFeedScreen
#:import MainScreen view.screens.main.MainScreen
#:import SettingsScreen view.screens.main.SettingsScreen
#:import TestingScreen view.screens.main.TestingScreen

#:import ColorsScreen view.screens.settings.ColorsScreen
#:import HeightScreen view.screens.settings.HeightScreen
#:import NoteScreen view.screens.settings.NoteScreen
#:import OperatorScreen view.screens.settings.OperatorScreen
#:import PlotScreen view.screens.settings.PlotScreen
#:import SensorsScreen view.screens.settings.SensorsScreen
#:import UpdateScreen view.screens.settings.UpdateScreen

#:import NewNoteScreen view.screens.settings.NewNoteScreen

#:import CalibrateScreen view.screens.settings.CalibrateScreen
#:import CalibratePointScreen view.screens.settings.CalibratePointScreen

#:import TestInProgressScreen view.screens.main.testing.TestInProgressScreen
#:import TestingResultsScreen view.screens.main.testing.TestingResultsScreen
#:import BreakHeightScreen view.screens.main.testing.BreakHeightScreen
#:import TestsScreen view.screens.main.testing.TestsScreen

<GranuScreenManager>:
    # Properties
    current: 'main_screen' # Start with the main screen

    # Root
    ExitScreen:
    LiveFeedScreen:
    MainScreen:
    SettingsScreen:
    TestingScreen:

    # Settings
    ColorsScreen:
    HeightScreen:
    NoteScreen:
    OperatorScreen:
    PlotScreen:
    SensorsScreen:
    UpdateScreen:

    # Settings - Notes
    NewNoteScreen:

    # Settings - Sensors
    CalibrateScreen:
    CalibratePointScreen:

    # Testing
    TestInProgressScreen:
    TestingResultsScreen:
    BreakHeightScreen:
    TestsScreen:

```

```python
from kivy.config import Config as KivyConfig
from kivy.app import App
from kivy.uix.screenmanager import ScreenManager, FadeTransition

import configurator as config

# Kivy Configuration
KivyConfig.set('kivy', 'desktop', 0) # Disable OS-specific features for testing
KivyConfig.set('kivy', 'keyboard_mode', 'systemanddock') # Allow barcode scanner and
                                                         # on screen keyboard
KivyConfig.set('graphics', 'height', 480) # Set window size to be the same as touchscreen
KivyConfig.set('graphics', 'width', 800)  # (not used when fullscreen enabled)
CLOCK_TYPE = "default"
KivyConfig.set('kivy', 'kivy_clock', CLOCK_TYPE)
KivyConfig.set('graphics', 'maxfps', 250)
KivyConfig.write()

class GranuScreenManager(ScreenManager):
    pass

class MainApp(App):
    def build(self):
        sm = GranuScreenManager(transition=FadeTransition(duration=0.1))
        return sm

if __name__ == "__main__":
    config.load() # Load our own app preferences
    MainApp().run()

```

```
<MainScreenInfoBox@Label>:
    font_size: 25
    color: 0,0,0,1
    halign: 'center'
    valign: 'center'
    canvas.before:
        Color:
            rgba: .74,.74,.74,1
        Rectangle:
            pos: self.pos
            size: self.size

<MainScreen>:
    name: 'main_screen'
    GranuContainer:
        GranuSideArea:
            GranuSideButton:
                text: 'Settings'
                on_release:
                    root.move_to('settings_screen') # Move to settings screen
            GranuSideButton:
                text: 'Testing'
                on_release:
                    root.move_to('testing_screen') # Move to testing screen
            GranuSideButton:
                text: 'Live Feed'
                on_release:
                    root.move_to('live_feed_screen') # Move to liveFeed screen
            GranuSideButton:
                text: 'Exit'
                on_release:
                    root.move_to('exit_screen') # Move to exit screen
        GranuContent:
            GranuTitle:
                text: 'Main Menu'
            Label:
                id: device_name
                font_size: 40
                text: 'Stalk Strength\nDevice 2.0'
                halign: 'center'
            Label:
                id: warning_text
                color: (0.8, 0, 0, 1)
                size_hint_max_y: 15
                font_size: 15
            GridLayout:
                cols: 2
                spacing: 10
                MainScreenInfoBox:
                    id: temperature
                    text: 'Temperature: ' + root.temperature +  u'\N{DEGREE SIGN}' + 'F'
                MainScreenInfoBox:
                    id: humidity
                    text: 'Humidity: ' + root.humidity +  "%"
                MainScreenInfoBox:
                    id: location
                    text: 'Location: ' + root.location
                MainScreenInfoBox:
                    id: time
                    text: 'Time: ' + root.time
```

```python
"""
The main screen contains four buttons for navigation:
Settings, Testing, Live Feed, and Exit

It also shows environment data: Temperature, Humidity, Location, and Time.
"""

from kivy.lang import Builder
from kivy.properties import StringProperty
from kivy.clock import Clock
from view.BaseScreen import BaseScreen
import datetime

from Sensor import Sensor

Builder.load_file('view/screens/main/MainScreen.kv')

INTERVAL = .004

class MainScreen(BaseScreen):
    temperature = StringProperty("0")
    humidity = StringProperty("0")
    location = StringProperty("0.00,0.00")
    time = StringProperty("0")
    def on_pre_enter(self):
        self.test_time = 0
        self.event = Clock.schedule_interval(self.update_values, INTERVAL)
        self.sensor_man = Sensor()
        if self.sensor_man.REAL_DATA is False:
            self.ids['warning_text'].text = 'WARNING: Using fake data.  Check console for stack trace.'

    def update_values(self, obj):
        self.time = datetime.datetime.now().strftime("%I:%M:%S %p")

    def on_leave(self):
        self.event.cancel()

```

```python
from connections import *

def motor_test():
    print("\n *********Beginning Motor Test********** \n")
    duration = 1
    freq = 50

    for motor, in1, in2, pwm in zip(MOTORS, IN1, IN2, PWM):
        p = GPIO.PWM(pwm, freq)
        p.start(0)

        # Drive the motor clockwise
        print("Driving Motor {} clockwise for {} seconds".format(motor, duration))
        GPIO.output(in1, GPIO.HIGH)
        GPIO.output(in2, GPIO.LOW)
        p.ChangeDutyCycle(50)
        sleep(duration)

        # Drive the motor counterclockwise
        print("Driving Motor {} counterclockwise for {} seconds".format(motor, duration))
        GPIO.output(in1, GPIO.LOW)
        GPIO.output(in2, GPIO.HIGH)
        p.ChangeDutyCycle(100)
        sleep(duration)

        # Reset all the GPIO pins by setting them to LOW
        GPIO.output(in1, GPIO.LOW)
        GPIO.output(in2, GPIO.LOW)
        p.stop()

    print("\n *********Ending Motor Test********** \n")

if __name__ == "__main__":
    motor_test()

```

```
<NewNoteScreen>:
    name: 'new_note_screen'
    GranuContainer:
        GranuSideArea:
            GranuSideButton:
                text: 'Save'
                on_release:
                    if root.save(): root.back()
            GranuSideButton:
                text: 'Cancel'
                on_release:
                    root.back()
        GranuContent:
            GranuTitle:
                text: 'New Note'
            GridLayout:
                rows: 1
                cols: 1
                StrInput:
                    id: note
                    font_size: 30
                    multiline: False
                    size_hint_max_y: 30+15

```

```python
"""
An input text box that, when selected, allows the user to type in a new note via a touch
screen keyboard that will pop up. The input text box will iniinputally be empty.
"""

from kivy.lang import Builder

import configurator as config
from view.BaseScreen import BaseScreen
from view.input.StrInput import StrInput

Builder.load_file('view/screens/settings/NewNoteScreen.kv')

class NewNoteScreen(BaseScreen):
    def on_pre_enter(self):
        input = self.ids['note']
        input.text = ''

    def on_enter(self):
        """Once the Screen loads, focus the Texinputnput"""
        input = self.ids['note']
        input.focus = True

    def save(self):
        notes = config.get('notes', {
            "pretest": [],
            "posttest": [],
            "bank": []
        })
        input = self.ids['note']

        note = input.text
        valid = input.validate()
        exists = (note in notes['pretest']) or (note in notes['posttest']) \
            or (note in notes['bank'])

        if valid and not exists:
            notes['bank'].append(input.text)
            config.set('notes', notes)
            return True
        else:
            input.show_invalid()
            input.focus = True
            return False

```

```
<Note>
    canvas.before:
        Color:
            rgba: (.0, 0.9, .1, .3) if self.selected else (0, 0, 0, 1)
        Rectangle:
            pos: self.pos
            size: self.size

<NoteList>
    viewclass: 'Note'
    SelectableRecycleBoxLayout:

<ListTitle@Label>
    size_hint_y: None
    height: self.font_size
    font_size: 15

<MoveButton@Button>
    size_hint_y: None
    height: self.font_size + 30
    font_size: 15

<NoteScreen>:
    name: 'note_screen'
    GranuContainer:
        GranuSideArea:
            id: note_buttons
        GranuContent:
            GranuTitle:
                text: 'Notes'
            BoxLayout:
                orientation: 'horizontal'
                size_hint_y: None
                height: self.minimum_height
                ListTitle:
                    text: 'Pre-test Notes'
                ListTitle:
                    text: 'Post-test Notes'
                ListTitle:
                    text: 'Note Bank'
            BoxLayout:
                orientation: 'horizontal'
                NoteList:
                    id: pretest
                    on_interaction:
                        posttest.clear_selection()
                        bank.clear_selection()
                        root.test_buttons()
                    on_deselect_all:
                        root.default_buttons()
                NoteList:
                    id: posttest
                    on_interaction:
                        pretest.clear_selection()
                        bank.clear_selection()
                        root.test_buttons()
                    on_deselect_all:
                        root.default_buttons()
                NoteList:
                    canvas.after:
                        Color:
                            rgba: 0.5, 0.5, 0.5, 1
                        Line:
                            width: 1
                            rectangle: self.x, self.y, 1, self.height
                    id: bank
                    on_interaction:
                        pretest.clear_selection()
                        posttest.clear_selection()
                        root.bank_buttons()
                    on_deselect_all:
                        root.default_buttons()
            BoxLayout:
                orientation: 'horizontal'
                size_hint_y: None
                height: self.minimum_height
                MoveButton:
                    text: 'Move Here'
                    on_release:
                        root.move_to_pretest()
                        root.default_buttons()
                MoveButton:
                    text: 'Move Here'
                    on_release:
                        root.move_to_posttest()
                        root.default_buttons()
                MoveButton:
                    text: 'Move Here'
                    on_release:
                        root.move_to_bank()
                        root.default_buttons()
            AnchorLayout:
                anchor_x: 'right'
                anchor_y: 'center'
                size_hint_y: None
                height: clear_button.height - 6
                Button:
                    id: clear_button
                    text: 'Clear Selection'
                    size_hint_y: None
                    height: self.font_size + 40
                    size_hint_x: 0.25
                    on_release:
                        pretest.clear_selection()
                        posttest.clear_selection()
                        bank.clear_selection()
                        root.default_buttons()

```

```python
from kivy.lang import Builder
from kivy.properties import ListProperty

import configurator as config
from view.BaseScreen import BaseScreen
from view.SelectableList import SelectableList, SelectableListBehavior, SelectableRecycleBoxLayout
from view.elements import *

Builder.load_file('view/screens/settings/NoteScreen.kv')

class Note(SelectableListBehavior, Label):
    pass

class NoteList(SelectableList):
    def update(self, k, val):
        self.data = [{'text': str(x)} for x in self.list_data]

class NoteScreen(BaseScreen):
    '''Manages the Notes.

    Be careful not to make a shallow copy of list_data for any SelectableList'''

    # Setup!

    def __init__(self, **kwargs):
        '''Creates Kivy Buttons to be able to dynamically change the sidebar actions
        based on interaction with the lists.'''
        super(BaseScreen, self).__init__(**kwargs)

        self.save_button = GranuSideButton(text = 'Save')
        self.save_button.bind(on_release = self.save)
        self.new_button = GranuSideButton(text = 'New')
        self.new_button.bind(on_release = self.new)
        self.remove_button = GranuSideButton(text = 'Remove')
        self.remove_button.bind(on_release = self.remove)
        self.delete_button = GranuSideButton(text = 'Delete')
        self.delete_button.bind(on_release = self.delete)

    def on_pre_enter(self):
        """Before the Screen loads, read the configuration file to get the current
        list of notes. Show the default buttons."""

        # Get notes from config file
        notes = config.get('notes', {
            "pretest": [],
            "posttest": [],
            "bank": []
        })
        # Set the data
        self.ids['pretest'].list_data = notes["pretest"]
        self.ids['posttest'].list_data = notes["posttest"]
        self.ids['bank'].list_data = notes["bank"]

        # Add Buttons
        self.default_buttons()

    def _save_config(self):
        '''Save the notes to the configuration file.'''
        config.set('notes', {
            "pretest": self.ids['pretest'].list_data,
            "posttest": self.ids['posttest'].list_data,
            "bank": self.ids['bank'].list_data
        })


    # Button Changes

    def default_buttons(self):
        buttons = self.ids['note_buttons']
        buttons.clear_widgets()
        buttons.add_widget(self.save_button)
        buttons.add_widget(self.new_button)
        buttons.add_widget(Widget())
        buttons.add_widget(Widget())

    def test_buttons(self):
        buttons = self.ids['note_buttons']
        buttons.clear_widgets()
        buttons.add_widget(self.save_button)
        buttons.add_widget(self.new_button)
        buttons.add_widget(Widget())
        buttons.add_widget(self.remove_button)

    def bank_buttons(self):
        buttons = self.ids['note_buttons']
        buttons.clear_widgets()
        buttons.add_widget(self.save_button)
        buttons.add_widget(self.new_button)
        buttons.add_widget(Widget())
        buttons.add_widget(self.delete_button)

    # Button Actions

    def save(self, obj):
        self._save_config()
        super(NoteScreen, self).back()

    def new(self, obj):
        self._save_config()
        self.move_to('new_note_screen')

    def remove(self, obj):
        '''Move selected notes to the Note Bank'''
        notes = self.ids['pretest'].remove_selected() \
            + self.ids['posttest'].remove_selected()
        self.ids['bank'].add_items(notes)
        self.default_buttons()

    def delete(self, obj):
        self.ids['bank'].remove_selected()
        self.default_buttons()

    # Content Buttons

    def move_to_pretest(self):
        notes = self.ids['pretest'].remove_selected() \
            + self.ids['posttest'].remove_selected() \
            + self.ids['bank'].remove_selected()
        self.ids['pretest'].add_items(notes)

    def move_to_posttest(self):
        notes = self.ids['pretest'].remove_selected() \
            + self.ids['posttest'].remove_selected() \
            + self.ids['bank'].remove_selected()
        self.ids['posttest'].add_items(notes)

    def move_to_bank(self):
        notes = self.ids['pretest'].remove_selected() \
            + self.ids['posttest'].remove_selected() \
            + self.ids['bank'].remove_selected()
        self.ids['bank'].add_items(notes)
```

```json
{
    "title" : "Numeric",
    "description" : "A numeric keypad",
    "cols" : 3,
    "rows": 4,
    "normal_1": [
    ["7", "7", "7", 1],
    ["8", "8", "8", 1],
    ["9", "9", "9", 1]],
    "normal_2": [
    ["4", "4", "4", 1],
    ["5", "5", "5", 1],
    ["6", "6", "6", 1]],
    "normal_3": [
    ["1", "1", "1", 1],
    ["2", "2", "2", 1],
    ["3", "3", "3", 1]],
    "normal_4": [
    ["0", "0", "0", 1],
    [".", ".", ".", 1],
    ["\u232b", null, "backspace", 1]],
    "shift_1": [
    ["7", "7", "7", 1],
    ["8", "8", "8", 1],
    ["9", "9", "9", 1]],
    "shift_2": [
    ["4", "4", "4", 1],
    ["5", "5", "5", 1],
    ["6", "6", "6", 1]],
    "shift_3": [
    ["1", "1", "1", 1],
    ["2", "2", "2", 1],
    ["3", "3", "3", 1]],
    "shift_4": [
    ["0", "0", "0", 1],
    [".", ".", ".", 1],
    ["\u232b", null, "backspace", 1]]
}
```

```
<OperatorScreen>:
    name: 'operator_screen'
    GranuContainer:
        GranuSideArea:
            GranuSideButton:
                text: 'Save'
                on_release:
                    if root.save(): root.back()
            GranuSideButton:
                text: 'Cancel'
                on_release:
                    root.back()
            GranuNone:
            GranuNone:
        GranuContent:
            GranuTitle:
                text: 'Operator'
            GridLayout:
                rows: 1
                cols: 1
                StrInput:
                    id: operator
                    font_size: 30
                    multiline: False
                    size_hint_max_y: 30+15

```

```python
"""
An input text box that, when selected, allows the user to type in the current Operator
setting via a touch screen keyboard that will pop up. The value in the input text box
when you first visit this view is whatever value for the Operator setting is currently
stored in our settings file .
"""

from kivy.lang import Builder

import configurator as config
from view.BaseScreen import BaseScreen
from view.input.StrInput import StrInput

Builder.load_file('view/screens/settings/OperatorScreen.kv')

class OperatorScreen(BaseScreen):
    def on_pre_enter(self):
        """Before the Screen loads, read the configuration file to get the current
        operator and set the TextInput text."""
        input = self.ids['operator']
        input.text = str(config.get('operator', "Default User"))
        input.validate()

    def on_enter(self):
        """Once the Screen loads, focus the TextInput"""
        input = self.ids['operator']
        input.focus = True
        input.select_all()

    def save(self):
        """Save button was pressed: save the new operator in the configuration file."""
        input = self.ids['operator']
        valid = input.validate()
        if valid:
            config.set('operator', str(input.text))
            return True
        else:
            input.focus = True
            return False

```

```
<PlotScreen>:
    name: 'plot_screen'
    GranuContainer:
        GranuSideArea:
            GranuSideButton:
                text: 'Save'
                on_release:
                    if root.save(): root.back()
            GranuSideButton:
                text: 'Cancel'
                on_release:
                    root.back()
        GranuContent:
            GranuTitle:
                text: 'Plot Number'
            StackLayout:
                padding: 20
                IntInput:
                    id: plot_num
                    font_size: 30
                    size_hint_max_y: 30+15
```

```python
"""
An input text box that, when selected, allows the user to type in the current Plot
setting via a touch screen number pad that will pop up. The value in the input text box
when you first visit this view is whatever value for the Plot setting is currently stored
in our settings file.
"""

from kivy.lang import Builder

import configurator as config
from view.BaseScreen import BaseScreen
from view.input.IntInput import IntInput

Builder.load_file('view/screens/settings/PlotScreen.kv')

class PlotScreen(BaseScreen):
    def on_pre_enter(self):
        """Before the Screen loads, read the configuration file to get the current
        plot number."""
        input = self.ids['plot_num']
        input.text = str(config.get('plot_num', 1))
        input.validate()

    def on_enter(self):
        """Once the Screen loads, focus the TextInput"""
        input = self.ids['plot_num']
        input.focus = True
        input.select_all()

    def save(self):
        """Save button was pressed: save the new height in the configuration file.
        Returns True if save was successful.  False otherwise."""
        input = self.ids['plot_num']
        valid = input.validate()
        if valid:
            config.set('plot_num', int(input.text))
            return True
        else:
            input.focus = True
            return False
```

```python
from .connections import *

class Pot:

    def __init__(self):
        self.pot = 0.0

    def get_data(self):
        try:
            self.pot = POT_CHAN.value
            return self.pot
        except:
            return self.pot

```

Plant Stalk Measurement Device Software

To run the software, you must have python3 and Kivy installed.

Navigate to the src/ folder in terminal and run:

```
python3 main.py
```

Compiling the Docmentation as PDF

- Download Doxygen and open Doxyfile and run
  - Make sure to ouput LaTeX files "as an intermediate format for [hyperlinked] PDF"
- Download LaTeX https://www.latex-project.org/get/
- Add LaTeX's binaries and scripts to your PATH
- Run the make file in the LaTeX folder to generate a PDF

```
<SelectableRecycleBoxLayout>:
    default_size: None, dp(56)
    default_size_hint: 1, None
    size_hint_y: None
    height: self.minimum_height
    orientation: 'vertical'
    multiselect: True
    touch_multiselect: True
    keyboard_mode: 'managed'

```

```python
from kivy.lang import Builder

from kivy.uix.recycleview import RecycleView
from kivy.uix.recycleview.views import RecycleDataViewBehavior
from kivy.uix.label import Label
from kivy.properties import BooleanProperty
from kivy.uix.recycleboxlayout import RecycleBoxLayout
from kivy.uix.behaviors import FocusBehavior
from kivy.uix.recycleview.layout import LayoutSelectionBehavior
from kivy.uix.behaviors import ButtonBehavior

from kivy.properties import ListProperty

Builder.load_file('view/SelectableList.kv')

class SelectableListBehavior(RecycleDataViewBehavior):
    '''Add selection support to a Label.'''
    index = None
    selected = BooleanProperty(False)
    selectable = BooleanProperty(True)

    def refresh_view_attrs(self, rv, index, data):
        '''Catch and handle data changes.'''
        self.index = index
        return super(SelectableListBehavior, self).refresh_view_attrs(rv, index, data)

    def on_touch_down(self, touch):
        '''Select this item on touch down.'''
        if super(SelectableListBehavior, self).on_touch_down(touch):
            return True
        if self.collide_point(*touch.pos) and self.selectable:
            ret = self.parent.select_with_touch(self.index, touch)
            self.parent.parent._interact() # SelectableList was interacted with
            return ret

    def apply_selection(self, rv, index, is_selected):
        '''Respond to the selection of items in the view.'''
        self.selected = is_selected

class SelectableRecycleBoxLayout(FocusBehavior, LayoutSelectionBehavior,
                                 RecycleBoxLayout):
    ''' Adds selection and focus behaviour to the view. '''

class SelectableList(RecycleView):
    '''A selectable list widget, that allows you to modify its data.

    Known issues: Changing data in a RecycleView causes "sys.excepthook' errors at close:
    https://github.com/kivy/kivy/issues/5986'''

    __events__ = ('on_interaction', 'on_deselect_all', ) # Add an event that can be defined in a kv file
    list_data = ListProperty() # List of strings to be shown in the list

    def __init__(self, **kwargs):
        '''Update the SelectableList's RecycleView data whenever list_data changes.'''
        super(SelectableList, self).__init__(**kwargs)
        self.bind(list_data=self.update)

    # Abstract!
    def update(self, k, val):
        '''Uses list_data to generate SelectableListWidgets.'''

    def _interact(self):
        '''Called by a list item when it is touched. Dispatches an on_interaction
        event.'''
        self.dispatch('on_interaction') # Call the Kivy event on_interaction
        if len(self.layout_manager.selected_nodes) == 0:
            self.dispatch('on_deselect_all')

    def clear_selection(self):
        '''Clears the selection in the SelectableList'''
        lm = self.layout_manager
        lm.clear_selection()

    def get_selected(self):
        '''Returns the items currently selected.'''
        lm = self.layout_manager
        sels = []
        for i in lm.selected_nodes:
            sels.append(self.list_data[i])
        return sels

    def remove_selected(self):
        '''Removes the selection in the SelectableList.  Returns the items removed.'''
        lm = self.layout_manager
        removed = []
        for i in lm.selected_nodes:
            removed.append(self.list_data[i])
        lm.clear_selection()
        for item in removed:
            self.list_data.remove(item)
        return removed

    def add_items(self, list):
        '''Add items to the list.'''
        self.list_data = self.list_data + list

    def on_interaction(self, *largs):
        '''A Kivy event: can be defined in a kv file.  Called when the SelectableList
        gains focus.'''
        pass

    def on_deselect_all(self, *largs):
        '''A Kivy event: can be defined in a kv file.  Called when the user deselects the
        last selected item in the SelectableList are deselected.'''
        pass

```

```python
import traceback
import sys
try:
    from .SensorReal import Sensor
except:
    print(traceback.print_exc()) # Tell us what happened
    from .SensorFake import Sensor

```

```python


class Sensor:
    def __init__(self):
        self.REAL_DATA = False
        self.keys = ["Temperature","Humidity","Location","X Load","Y Load","Pot Angle","IMU Angle"]
        self.sensor_data = {}
        self.temp_fake = 0
        self.hum_fake = 0
        self.loc_fake = 0
        self.x_fake = 0
        self.y_fake = 0
        self.pot_fake = 0
        self.imu_fake = 0

    def get_header_data(self):
        self.temp_fake += 1
        self.hum_fake += 2
        self.loc_fake += 4
        self.sensor_data["Temperature"] = self.temp_fake
        self.sensor_data["Humidity"] = self.hum_fake
        self.sensor_data["Location"] = self.loc_fake
        return self.sensor_data

    def get_sensor_data(self):
        self.x_fake += 8
        self.y_fake += 16
        self.pot_fake += 32
        self.imu_fake += 64
        self.sensor_data["X Load"] = self.x_fake
        self.sensor_data["Y Load"] = self.y_fake
        self.sensor_data["Pot Angle"] = self.pot_fake
        self.sensor_data["IMU Angle"] = self.imu_fake
        return self.sensor_data

    def get_all_data(self):
        self.temp_fake += 1
        self.hum_fake += 2
        self.loc_fake += 4
        self.x_fake += 8
        self.y_fake += 16
        self.pot_fake += 32
        self.imu_fake += 64
        self.sensor_data["Temperature"] = self.temp_fake
        self.sensor_data["Humidity"] = self.hum_fake
        self.sensor_data["Location"] = self.loc_fake
        self.sensor_data["X Load"] = self.x_fake
        self.sensor_data["Y Load"] = self.y_fake
        self.sensor_data["Pot Angle"] = self.pot_fake
        self.sensor_data["IMU Angle"] = self.imu_fake
        return self.sensor_data

    def get_sensor_keys(self):
        return self.keys
```

```python
from sensors.Temperature import Temperature
from sensors.Humidity import Humidity
from sensors.Location import Location
from sensors.X_Load import X_Load
from sensors.Y_Load import Y_Load
from sensors.Pot import Pot
from sensors.IMU import IMU
import datetime
import board
import busio
import adafruit_ads1x15.ads1015 as ADS
from adafruit_ads1x15.analog_in import AnalogIn

class Sensor:

    def __init__(self):
        self.REAL_DATA = True
        self.keys = ["Temperature","Humidity","Location","X Load","Y Load","Pot Angle","IMU Angle"]
        self.temp = Temperature()
        self.hum = Humidity()
        self.location = Location()
        self.x_load = X_Load()
        self.y_load = Y_Load()
        self.pot_angle = Pot()
        self.imu_angle = IMU()
        self.sensor_data = {}
        self.temp_fake = 0
        self.hum_fake = 0
        self.loc_fake = 0
        self.x_fake = 0
        self.y_fake = 0
        self.pot_fake = 0
        self.imu_fake = 0

    def get_header_data(self):
        self.sensor_data["Temperature"] = self.temp.get_data()
        self.sensor_data["Humidity"] = self.hum.get_data()
        self.sensor_data["Location"] = self.location.get_data()
        return self.sensor_data

    def get_sensor_data(self):
        self.sensor_data["X Load"] = round(self.x_load.get_data(),4)
        self.sensor_data["Y Load"] = round(self.y_load.get_data(),4)
        self.sensor_data["Pot Angle"] = round(self.pot_angle.get_data(),3)
        self.sensor_data["IMU Angle"] = round(self.imu_angle.get_data(),3)
        return self.sensor_data

    def get_all_data(self):
        self.sensor_data["Temperature"] = self.temp.get_data()
        self.sensor_data["Humidity"] = self.hum.get_data()
        self.sensor_data["Location"] = self.location.get_data()
        self.sensor_data["X Load"] = round(self.x_load.get_data(),4)
        self.sensor_data["Y Load"] = round(self.y_load.get_data(),4)
        self.sensor_data["Pot Angle"] = round(self.pot_angle.get_data(),3)
        self.sensor_data["IMU Angle"] = round(self.imu_angle.get_data(),3)
        return self.sensor_data

    def get_sensor_keys(self):
        return self.keys

if __name__ == "__main__":
    sensor = Sensor()
    print("\n *********Beginning Sensor Test********** \n")
    print("Sensor Data: ")
    data_array = sensor.get_sensor_data()
    for key in sensor.get_sensor_keys():
        print(key, data_array[key])
    print("\n **********Ending Sensor Test********** \n")

```

SensorsScreen.kv
Printed: 4/13/19, 12:03:03 AM

```
<SensorButton>:
    font_size: 30
    halign: 'center'

<SensorsScreen>:
    name: 'sensors_screen'
    GranuContainer:
        GranuSideArea:
            GranuSideButton:
                text: 'Back'
                on_release:
                    root.back()
        GranuContent:
            GranuTitle:
                text: 'Sensor Calibration'
            GridLayout:
                id: sensor_list
                cols: 3
                rows: 3
                spacing: 10

```

```python
from kivy.lang import Builder
from kivy.clock import Clock
from kivy.uix.button import Button
from kivy.properties import ObjectProperty

from view.BaseScreen import BaseScreen
from Sensor import Sensor

Builder.load_file('view/screens/settings/SensorsScreen.kv')

class SensorButton(Button):
    def __init__(self, name, parent_screen, calib_screen, **kwargs):
        '''Set the sensor name (which is also the key to retrieve calibration settings
        from the config file.  Reference the parent screen for move_to() function.
        Reference the calibration screen to set it up to calibrate this setting.'''
        super(Button, self).__init__(**kwargs)
        self.name = name
        # Screen References
        self.parent_screen = parent_screen
        self.calib_screen = calib_screen
        # Kivy Properties
        self.text = name

    def on_release(self):
        self.calib_screen.set_sensor(self.name)
        self.parent_screen.move_to('calibrate_screen')

class SensorsScreen(BaseScreen):
    def __init__(self, **kwargs):
        '''Add a button for each sensor.'''
        super(BaseScreen, self).__init__(**kwargs)
        self.senseMan = Sensor()
        def gui_init(dt):
            '''Called once the Kivy file is parsed. Needed so we can access Kivy IDs.'''
            calib_screen = self.manager.get_screen('calibrate_screen')
            for s in self.senseMan.get_sensor_keys():
                # Perhaps Location and Time should be accessed in some other way?
                if s=='Location' or s=='Time': continue
                # Sensor name, parent screen (of button), calibration screen
                self.ids['sensor_list'].add_widget(SensorButton(s, self, calib_screen))
        Clock.schedule_once(gui_init)

```

```
#:import Path pathlib.Path

<LoadDialog>:
    title: 'Load Settings'
    title_align: 'center'
    size_hint: (0.9, 0.9)
    BoxLayout:
        orientation: "vertical"
        FileChooserIconView:
            id: filechooser
            path: str(Path.home())
        BoxLayout:
            size_hint_y: None
            height: 30
            Button:
                text: "Cancel"
                on_release: root.cancel()
            Button:
                text: "Load"
                on_release: root.load(filechooser.path, filechooser.selection[0])

<SaveDialog>:
    title: 'Save Settings'
    title_align: 'center'
    size_hint: (0.9, 0.9)
    BoxLayout:
        orientation: "vertical"
        BoxLayout:
            orientation: "horizontal"
            size_hint_y: None
            height: filename.height
            Label:
                text: 'Filename:'
                size_hint_x: None
            StrInput:
                id: filename
                text: 'settings.json'
                size_hint_y: None
                height: 30
                multiline: False
        FileChooserIconView:
            id: filechooser
            path: str(Path.home())
            on_selection: filename.text = self.selection and self.selection[0] or ''
        BoxLayout:
            size_hint_y: None
            height: 30
            Button:
                text: "Cancel"
                on_release: root.cancel()
            Button:
                text: "Save"
                on_release: root.save(filechooser.path, filename.text)

<SettingsButton@Button>:
    font_size: 30
    halign: 'center'

<SettingsScreen>:
    name: 'settings_screen'
    GranuContainer:
        GranuSideArea:
            GranuNone:
            GranuNone:
            GranuNone:
            GranuSideButton:
                text: 'Back'
                on_release:
                    root.manager.transition.direction = 'right'
                    root.move_to('main_screen') # Move to main screen
        GranuContent:
            GranuTitle:
                text: 'Settings'
            GridLayout:
                cols: 3
                rows: 3
                spacing: 10
                row_default_height: self.height/4
                SettingsButton:
                    text: 'Height'
                    on_release:
                        root.move_to('height_screen')
                SettingsButton:
                    text: 'Plot'
                    on_release:
                        root.manager.transition.direction = 'left'
                        root.move_to('plot_screen')
                SettingsButton:
                    text: 'Notes'
                    on_release:
                        root.manager.transition.direction = 'left'
                        root.move_to('note_screen')
                SettingsButton:
                    text: 'Sensors'
                    on_release:
                        root.manager.transition.direction = 'left'
                        root.move_to('sensors_screen')
                SettingsButton:
                    text: 'Operator'
                    on_release:
                        root.manager.transition.direction = 'left'
                        root.move_to('operator_screen')
            StackLayout:
                orientation: 'rl-bt'
                size_hint_y: None
                spacing: 5
                height: load_button.height - 6
                Button:
                    id: load_button
                    text: 'Load File'
                    size_hint_y: None
                    height: self.font_size + 40
                    size_hint_x: 0.25
                    on_release: root.show_load()
                Button:
                    text: 'Save File'
                    size_hint_y: None
                    height: self.font_size + 40
                    size_hint_x: 0.25
                    on_release: root.show_save()

```

```python
"""
From the settings screen you can navigate to these options: Height, Plot, Operator,
Folder, Notes
"""

import os

from kivy.lang import Builder
from kivy.uix.floatlayout import FloatLayout
from kivy.properties import ObjectProperty
from kivy.properties import StringProperty
from kivy.uix.popup import Popup

import configurator as config
from view.BaseScreen import BaseScreen

Builder.load_file('view/screens/main/SettingsScreen.kv')

class LoadDialog(Popup):
    '''A dialog to load a file.  The load and cancel properties point to the
    functions called when the load or cancel buttons are pressed.'''
    load = ObjectProperty(None)
    cancel = ObjectProperty(None)


class SaveDialog(Popup):
    '''A dialog to save a file.  The save and cancel properties point to the
    functions called when the save or cancel buttons are pressed.'''
    save = ObjectProperty(None)
    cancel = ObjectProperty(None)

class SettingsScreen(BaseScreen):
    def dismiss_popup(self):
        self._popup.dismiss()

    def show_load(self):
        self._popup = LoadDialog(load=self.load, cancel=self.dismiss_popup)
        self._popup.open()

    def show_save(self):
        self._popup = SaveDialog(save=self.save, cancel=self.dismiss_popup)
        self._popup.open()

    def load(self, path, filename):
        config.load_from(os.path.join(path, filename))
        self.dismiss_popup()

    def save(self, path, filename):
        config.save_as(os.path.join(path, filename))
        self.dismiss_popup()

```

```
<SingleSelectableRecycleBoxLayout>:
    default_size: None, dp(56)
    default_size_hint: 1, None
    size_hint_y: None
    height: self.minimum_height
    orientation: 'vertical'

    keyboard_mode: 'managed'

```

```python
from kivy.lang import Builder

from kivy.uix.recycleview import RecycleView
from kivy.uix.recycleview.views import RecycleDataViewBehavior
from kivy.uix.label import Label
from kivy.properties import BooleanProperty
from kivy.uix.recycleboxlayout import RecycleBoxLayout
from kivy.uix.behaviors import FocusBehavior
from kivy.uix.recycleview.layout import LayoutSelectionBehavior
from kivy.uix.behaviors import ButtonBehavior

from kivy.properties import ListProperty

Builder.load_file('view/SingleSelectableList.kv')

class SingleSelectableListBehavior(RecycleDataViewBehavior):
    '''Add selection support to a Label.'''
    index = None
    selected = BooleanProperty(False)
    singleSelectable = BooleanProperty(True)

    def refresh_view_attrs(self, rv, index, data):
        '''Catch and handle data changes.'''
        self.index = index
        return super(SingleSelectableListBehavior, self).refresh_view_attrs(rv, index, data)

    def on_touch_down(self, touch):
        '''Select this item on touch down.'''
        if super(SingleSelectableListBehavior, self).on_touch_down(touch):
            return True
        if self.collide_point(*touch.pos) and self.selectable:
            ret = self.parent.select_with_touch(self.index, touch)
            self.parent.parent._interact() # SelectableList was interacted with
            return ret

    def apply_selection(self, rv, index, is_selected):
        '''Respond to the selection of items in the view.'''
        self.selected = is_selected

class SingleSelectableRecycleBoxLayout(FocusBehavior, LayoutSelectionBehavior,
                                 RecycleBoxLayout):
    ''' Adds selection and focus behaviour to the view. '''

class SingleSelectableList(RecycleView):
    '''A selectable list widget, that allows you to modify its data.

    Known issues: Changing data in a RecycleView causes "sys.excepthook' errors at close:
    https://github.com/kivy/kivy/issues/5986'''

    __events__ = ('on_interaction', 'on_deselect_all', ) # Add an event that can be defined in a kv file
    list_data = ListProperty() # List of strings to be shown in the list

    def __init__(self, **kwargs):
        '''Update the SelectableList's RecycleView data whenever list_data changes.'''
        super(SingleSelectableList, self).__init__(**kwargs)
        self.bind(list_data=self.update)

    # Abstract!
    def update(self, k, val):
        '''Uses list_data to generate SelectableListWidgets.'''

    def _interact(self):
        '''Called by a list item when it is touched. Dispatches an on_interaction
        event.'''
        self.dispatch('on_interaction') # Call the Kivy event on_interaction
        if len(self.layout_manager.selected_nodes) == 0:
            self.dispatch('on_deselect_all')

    def clear_selection(self):
        '''Clears the selection in the SelectableList'''
        lm = self.layout_manager
        lm.clear_selection()

    def get_selected(self):
        '''Returns the items currently selected.'''
        lm = self.layout_manager
        sels = []
        for i in lm.selected_nodes:
            sels.append(self.list_data[i])
        return sels

    def remove_selected(self):
        '''Removes the selection in the SelectableList.  Returns the items removed.'''
        lm = self.layout_manager
        removed = []
        for i in lm.selected_nodes:
            removed.append(self.list_data[i])
        lm.clear_selection()
        for item in removed:
            self.list_data.remove(item)
        return removed

    def add_items(self, list):
        '''Add items to the list.'''
        self.list_data = self.list_data + list

    def on_interaction(self, *largs):
        '''A Kivy event: can be defined in a kv file.  Called when the SelectableList
        gains focus.'''
        pass

    def on_deselect_all(self, *largs):
        '''A Kivy event: can be defined in a kv file.  Called when the user deselects the
        last selected item in the SelectableList are deselected.'''
        pass

```

```
<StaticRecycleBoxLayout>:
    default_size: None, dp(30)
    default_size_hint: 1, None
    size_hint_y: None
    height: self.minimum_height
    orientation: 'vertical'
    multiselect: False
    touch_multiselect: False
    keyboard_mode: 'managed'

<StaticLabel>:

<StaticList>:
    viewclass: 'StaticLabel'
    StaticRecycleBoxLayout:

```

```python
from kivy.lang import Builder
from kivy.uix.recycleview import RecycleView
from kivy.uix.recycleview.views import RecycleDataViewBehavior
from kivy.uix.label import Label
from kivy.uix.recycleboxlayout import RecycleBoxLayout
from kivy.properties import ListProperty

Builder.load_file('view/StaticList.kv')

class StaticRecycleBoxLayout(RecycleBoxLayout):
    pass

class StaticLabel(RecycleDataViewBehavior, Label):
    '''Refresh Labels when list is changed.'''
    index = None

    def refresh_view_attrs(self, rv, index, data):
        '''Catch and handle data changes.'''
        self.index = index
        return super(StaticLabel, self).refresh_view_attrs(rv, index, data)

class StaticList(RecycleView):
    '''A static list widget.

    Known issues: Changing data in a RecycleView causes "sys.excepthook' errors at close:
    https://github.com/kivy/kivy/issues/5986'''

    list_data = ListProperty() # List of strings to be shown in the list

    def __init__(self, **kwargs):
        '''Update the StaticList's RecycleView data whenever list_data changes.'''
        super(StaticList, self).__init__(**kwargs)
        self.bind(list_data=self._update)

    def _update(self, k, val):
        '''Uses list_data to generate StaticLabels.'''
        self.data = [{'text': str(x)} for x in self.list_data]

```

```python
from kivy.clock import Clock
from kivy.uix.textinput import TextInput

import view.keyboard_man as km

class StrInput(TextInput):
    def validate(self):
        '''Make sure the string is not empty or all whitespace.'''
        notempty = len(self.text) > 0 # String is not empty or all whitespace
        notspace = not self.text.isspace()
        test = notempty and notspace
        if test:
            self.show_valid()
        else:
            self.show_invalid()
        return test

    def show_invalid(self):
        '''Colors the textinput red.'''
        self.background_color = (1, .7, .7, 1)

    def show_valid(self):
        '''Colors the textinput white.'''
        self.background_color = (1, 1, 1, 1)

    def on_text_validate(self):
        '''Called when enter is pressed.'''
        self.validate()
        Clock.schedule_once(self.focus_and_select)

    def on_focus(self, instance, value):
        '''When the StrInput is focused, show a qwerty keyboard.'''
        if value:
            km.show_keyboard(self, 'text')

    def focus_and_select(self, *args):
        '''Focus the TextInput and select all of its text.'''
        self.focus = True
        self.select_all()

```

```python
from connections import *

def temp_test():
    print("\n *********Beginning TEMP Test********** \n")

    for i in range(3):
        print("Temperature: ", am.temperature)
        print("Humidity: ", am.relative_humidity)
        time.sleep(2)

    print("\n *********Ending TEMP Test********** \n")

if __name__ == "__main__":
    temp_test()

```

```python
from .connections import *

class Temperature:

    def __init__(self):
        self.temp = 0.0

    def get_data(self):
        try:
            self.temp = am.temperature
            return self.temp
        except:
            return self.temp

```

```python
from tests.gpio_test import *
from tests.adc_test import *
from tests.gps_test import *
from tests.motor_test import *
from tests.temp_test import *
from tests.accl_test import *
import RPi.GPIO as GPIO

if __name__ == "__main__":
    temp_test()
    accl_test()
    gpio_test()
    adc_test()
    gps_test()
    motor_test()
    GPIO.cleanup()

```

```
#:import MeshLinePlot kivy.garden.graph.MeshLinePlot

<TestingResultsInfoBox@Label>:
    font_size: 25
    color: 0,0,0,1
    halign: 'center'
    valign: 'center'
    canvas.before:
        Color:
            rgba: .74,.74,.74,1
        Rectangle:
            pos: self.pos
            size: self.size

<TestingResultsScreen>:
    name: 'testing_results_screen'
    GranuContainer:
        GranuSideArea:
            GranuSideButton:
                text: 'Update\nNotes'
                on_release:
                    root.move_to('note_screen') # Move to note_screen height screen
            GranuSideButton:
                text: 'Break\nHeight'
                on_release:
                    root.move_to('break_height_screen') # Move to break height screen
            GranuSideButton:
                text: 'Reject'
                on_release:
                    root.move_to('main_screen') # Move to main screen
            GranuSideButton:
                text: 'Save'
                on_release:
                    root.save_test()
                    root.move_to('main_screen') # Move to main screen
        GranuContent:
            GranuTitle:
                text: 'Testing Results'
            Graph:
                id: graph_test
                plot: MeshLinePlot
                background_color: .5, .5, .5, 1
                xlabel:'Time(s)'
                ylabel:'X_Load(adc)'
                x_ticks_major:root.x_major
                y_ticks_major:root.y_major
                y_grid_label:True
                x_grid_label:True
                padding:5
                x_grid:True
                y_grid:True
                xmin:0
                ymin:0
                xmax:root.x_max
                ymax:root.y_max

```

```python
"""
Shows all data: Temperature, Humidity, Location, Time, and all Sensor data
"""

from kivy.lang import Builder
from kivy.properties import NumericProperty
from kivy.properties import StringProperty
from kivy.properties import ListProperty
from kivy.clock import Clock
from Sensor import Sensor
import datetime
import time
import math
from TestSingleton import TestSingleton

from view.BaseScreen import BaseScreen
from view.StaticList import StaticList
from view.elements import *
import configurator as config
import csv

from kivy.garden.graph import Graph, MeshLinePlot

Builder.load_file('view/screens/main/testing/TestingResultsScreen.kv')

ONE_SEC = 1


class TestingResultsScreen(BaseScreen):
    x_max = NumericProperty(1)
    y_max = NumericProperty(1)
    x_major = NumericProperty(1)
    y_major = NumericProperty(1)
    datasets = []


    def find_max_x_load(self):
        max = 0
        for dataset in self.datasets:
            if(dataset.x_load > max):
                max = dataset.x_load
        return max

    def on_enter(self):
        self.graph = self.ids['graph_test']
        self.plot = MeshLinePlot(color=[1, 1, 1, 1])
        ts = TestSingleton()
        self.datasets = ts.get_datasets()
        last_index = len(self.datasets) - 1

        self.x_max = math.ceil(self.datasets[last_index].timestamp / 5) * 5
        #self.y_max = math.ceil(self.find_max_x_load() / 10000) * 10000
        self.y_max = 2000
        self.x_major = int(self.x_max/5)
        self.y_major = int(self.y_max/5)


        self.plot.points = [(self.datasets[i].timestamp, self.datasets[i].pot_angle) for i in range(0, len(self.datasets))]
        #for i in range(0,len(self.datasets)):
        #    print("Time:",self.datasets[i].timestamp," -- X_Load:", self.datasets[i].x_load)

        self.graph.add_plot(self.plot)


    def save_test(self):
        ts = TestSingleton()
        self.datasets = ts.get_datasets()
        ts.set_break_height(str(config.get('break_height', 0)))

        #Prepare the notes
        notes = config.get('notes', {
            "pretest": [],
            "posttest": [],
            "bank": []
        })
        pre_notes = notes["pretest"]
        post_notes = notes["posttest"]
        while(len(pre_notes) < 5):
            pre_notes.append('')
        while(len(post_notes) < 5):
            post_notes.append('')
        dt = datetime.datetime.now()
        filename = 'Tests/' + dt.strftime('%Y_%m_%d_%H_%M_%S') + '.csv'

        with open(filename, 'w+', newline='') as csvFile:
            writer = csv.writer(csvFile)
            writer.writerow(['----------META DATA----------'])
            writer.writerow(['SOFTWARE VERSION', '2.0.0'])
            writer.writerow(['DEVICE OPERATOR', str(config.get('operator', 0))])
            writer.writerow(['----------TEST ATTRIBUTES----------'])
            writer.writerow(['FIELD', 'VALUE', 'UNIT'])
            writer.writerow(['YEAR', dt.strftime("%Y")])
            writer.writerow(['MONTH', dt.strftime("%m")])
            writer.writerow(['DAY', dt.strftime("%d")])
            writer.writerow(['TIME', dt.strftime("%H:%M:%S"), 'Local Time Zone'])
            writer.writerow(['PLOT', str(config.get('plot_num', 0)), '#'])
            writer.writerow(['HEIGHT', str(config.get('height', 0)), 'cm'])
            writer.writerow(['TEMPERATURE', '40', 'C'])
            writer.writerow(['HUMIDITY', '40', '%'])
            writer.writerow(['LATITUDE', '40', 'angular degrees'])
            writer.writerow(['LONGITUDE', '40', 'angular degrees'])
            writer.writerow(['----------OPTIONAL DATA----------'])
            writer.writerow(['PRE_TEST_NOTE_1', pre_notes[0]])
            writer.writerow(['PRE_TEST_NOTE_2', pre_notes[1]])
            writer.writerow(['PRE_TEST_NOTE_3', pre_notes[2]])
            writer.writerow(['PRE_TEST_NOTE_4', pre_notes[3]])
            writer.writerow(['PRE_TEST_NOTE_5', pre_notes[4]])
            writer.writerow(['POST_TEST_NOTE_1', post_notes[0]])
            writer.writerow(['POST_TEST_NOTE_2', post_notes[1]])
            writer.writerow(['POST_TEST_NOTE_3', post_notes[2]])
            writer.writerow(['POST_TEST_NOTE_4', post_notes[3]])
            writer.writerow(['POST_TEST_NOTE_5', post_notes[4]])
            writer.writerow(['BREAK_HEIGHT', str(config.get('break_height', 0)), 'cm'])
            writer.writerow(['LCA_WEIGTH', '0', 'g'])
            writer.writerow(['----------SENSOR CALIBRATION DATA (stored_value*A + B = raw_data)------'])
            writer.writerow(['SENSOR', 'A', 'B', 'UNIT', 'ID'])
            writer.writerow(['LOAD_X', '0', '0', 'N', 'loadx1'])
            writer.writerow(['LOAD_Y', '0', '0', 'Newton', 'loady1'])
            writer.writerow(['IMU', '0', '0', 'Deg', 'imu1'])
            writer.writerow(['POT', '0', '0', 'Deg', 'pot1'])
            writer.writerow(['TEMP', '0', '0', 'C', 'temp1'])
            writer.writerow(['HUM', '0', '0', '%', 'hum1'])
            writer.writerow(['----------TEST DATA-----------'])
            writer.writerow(['TIME (s)', 'ANGLE_POT', 'ANGLE_IMU', 'LOAD_X', 'LOAD_Y'])
            datasets = ts.get_datasets()
            for ds in datasets:
                writer.writerow([ds.timestamp, ds.pot_angle, ds.imu_angle, ds.x_load, ds.y_load])


        csvFile.close()

    def on_leave(self):
```

```python


        self.graph.remove_plot(self.plot)
        self.graph._clear_buffer()
```

```
<TestingInfoBox@Label>:
    font_size: 25
    color: 0,0,0,1
    halign: 'center'
    valign: 'center'
    canvas.before:
        Color:
            rgba: .74,.74,.74,1
        Rectangle:
            pos: self.pos
            size: self.size

<TestingScreen>:
    name: 'testing_screen'
    GranuContainer:
        GranuSideArea:
            GranuSideButton:
                text: 'Update\nNotes'
                on_release:
                    root.move_to('note_screen') # Move to notes screen
            GranuSideButton:
                text: "Start"
                on_release:
                    root.move_to('test_in_progress_screen') # Move to test in progress screen
            GranuSideButton:
                text: "Tests"
                on_release:
                    root.move_to('tests_screen') # Move to tests screen
            GranuSideButton:
                text: 'Back'
                on_release:
                    root.move_to('main_screen') # Move to main screen
        GranuContent:
            GranuTitle:
                text: 'Testing'
            GridLayout:
                cols: 2
                spacing: 10
                TestingInfoBox:
                    id: height
                    text: "Height: " + root.height_num + " cm"
                TestingInfoBox:
                    id: operator
                    text: "Operator: " + root.operator
                TestingInfoBox:
                    id: plot
                    text: "Plot: " + root.plot
                TestingInfoBox:
                    id: time
                    text: "Time: " + root.time
            BoxLayout:
                orientation: 'horizontal'
                size_hint_y: None
                height: self.minimum_height
                ListTitle:
                    text: 'Pre-test Notes'
                ListTitle:
                    text: 'Post-test Notes'
            BoxLayout:
                orientation: 'horizontal'
                StaticList:
                    id: pretest
                StaticList:
                    id: posttest


```

```python
"""
Testing Menu
"""

from kivy.lang import Builder
from kivy.properties import NumericProperty
from kivy.properties import StringProperty
from kivy.properties import ListProperty
from kivy.clock import Clock
from Sensor import Sensor

from view.BaseScreen import BaseScreen
from view.StaticList import StaticList
import configurator as config
from view.elements import *
import datetime

Builder.load_file('view/screens/main/TestingScreen.kv')

ONE_SEC = 1

class TestingScreen(BaseScreen):
    height_num = StringProperty("N/A")
    plot = StringProperty("N/A")
    operator = StringProperty("N/A")
    time = StringProperty("N/A")
    datasets = []

    def on_pre_enter(self):
        """Before the Screen loads, read the configuration file to get the current
        list of notes. Show the default buttons."""
        self.event = Clock.schedule_interval(self.update_time, ONE_SEC)
        self.height_num = str(config.get('height',0))
        self.plot = str(config.get('plot_num',0))
        self.operator = config.get('operator','N/A')
        self.time = datetime.datetime.now().strftime("%I:%M:%S %p")
        # Get notes from config file
        notes = config.get('notes', {
            "pretest": [],
            "posttest": [],
            "bank": []
        })
        # Set the data
        self.ids['pretest'].list_data = notes["pretest"]
        self.ids['posttest'].list_data = notes["posttest"]

    def update_time(self, obj):
        self.time = datetime.datetime.now().strftime("%I:%M:%S %p")

    def on_leave(self):
        self.event.cancel()
```

```
#:import MeshLinePlot kivy.garden.graph.MeshLinePlot

<TestInProgressInfoBox@Label>:
    font_size: 50
    color: 0,0,0,1
    halign: 'center'
    valign: 'center'
    color: 1,1,1,1
    canvas.before:
        Color:
            rgba: .5,.4,.4,1
        Rectangle:
            pos: self.pos
            size: self.size

<TestInProgressScreen>:
    name: 'test_in_progress_screen'
    GranuContainer:
        GranuSideArea:
            GranuNone:
            GranuNone:
            GranuNone:
            GranuSideButton:
                text: 'Stop'
                on_release:
                    root.move_to('testing_results_screen') # Move to testing results screen
        GranuContent:
            GranuTitle:
                text: 'Test in Progress'
            TestInProgressInfoBox:
                id: test_time_box
                text: "Test in progress for\n" + str(root.test_time) + " seconds"
            Graph:
                id: graph_test
                plot: MeshLinePlot
                background_color: .5, .5, .5, 1
                xlabel:'Time(s)'
                ylabel:'X_Load(adc)'
                x_ticks_major:root.x_major
                y_ticks_major:root.y_major
                y_grid_label:True
                x_grid_label:True
                padding:5
                x_grid:True
                y_grid:True
                xmin:0
                ymin:0
                xmax:root.x_max
                ymax:root.y_max
```

```python
"""
Test in Progress
"""

import datetime

from kivy.lang import Builder
from kivy.properties import NumericProperty
from kivy.properties import StringProperty
from kivy.properties import ListProperty
from kivy.clock import Clock
from Dataset import Dataset
from Sensor import Sensor
from TestSingleton import TestSingleton
from Sensor import Sensor

from view.BaseScreen import BaseScreen
import configurator as config
from kivy.config import Config as KivyConfig
from view.elements import *
import datetime
import time
import math

from kivy.garden.graph import Graph, MeshLinePlot

Builder.load_file('view/screens/main/testing/TestInProgressScreen.kv')

INTERVAL = .003
SECOND_CAP = 1/INTERVAL

class TestInProgressScreen(BaseScreen):
    test_time = NumericProperty(0)
    x_max = NumericProperty()
    y_max = NumericProperty()
    x_major = NumericProperty()
    y_major = NumericProperty()
    temperature = 0
    humidity = 0
    location = 0
    x_load = 0
    y_load = 0
    pot_angle = 0
    imu_angle = 0
    data_rate = 0
    second_counter = 0
    double_counter = 0
    start_time = 0
    start_timestamp = datetime.datetime.now().strftime("%I:%M:%S %p")

    def on_pre_enter(self):
        self.test_time = 0
        self.temperature = 0
        self.humidity = 0
        self.location = 0
        self.x_load = 0
        self.y_load = 0
        self.pot_angle = 0
        self.imu_angle = 0
        self.data_rate = 0
        self.second_counter = 0
        self.double_counter = 0
        self.start_time = datetime.datetime.now()
        self.datasets = []
        self.x_max = 5
        self.y_max = 2000
        self.x_major = int(self.x_max/5)
        self.y_major = int(self.y_max/5)
        self.datasets = []
        self.test_sensor = Sensor()
        self.plot = MeshLinePlot(color=[1, 1, 1, 1])

        self.event = Clock.schedule_interval(self.update_dataset, INTERVAL)
        #ClockBaseInterruptBehavior.interupt_next_only = True
    def find_max_x_load(self):
        max = 0
        for dataset in self.datasets:
            if(dataset.x_load > max):
                max = dataset.x_load
        return max

    def update_dataset(self, obj):
        self.second_counter += 1
        time_delta = datetime.datetime.now() - self.start_time
        total_time_passed = time_delta.seconds + (time_delta.microseconds * .000001)
        self.test_time = time_delta.seconds
        if self.second_counter >= SECOND_CAP/2:
            self.double_counter += 1
            self.second_counter = 0
            self.graph = self.ids['graph_test']
            self.graph.remove_plot(self.plot)
            self.graph._clear_buffer()
            self.plot = MeshLinePlot(color=[1, 1, 1, 1])
            last_index = len(self.datasets) - 1

            self.x_max = math.ceil(self.datasets[last_index].timestamp / 5) * 5
            #if(self.find_max_x_load() == 0):
             #   self.y_max = 10000
            #else:
            #    self.y_max = math.ceil(self.find_max_x_load() / 10000) * 10000
            self.x_major = int(self.x_max/5)
            #self.y_major = int(self.y_max/5)





            self.plot.points = [(self.datasets[i].timestamp, self.datasets[i].pot_angle) for i in range(0, len(self.datasets))]

            self.graph.add_plot(self.plot)



        sensor_values = self.test_sensor.get_sensor_data()
        self.x_load = sensor_values["X Load"]
        self.y_load = sensor_values["Y Load"]
        self.pot_angle = sensor_values["Pot Angle"]
        self.imu_angle = sensor_values["IMU Angle"]


        new_dataset = Dataset(total_time_passed, self.x_load, self.y_load, self.pot_angle, self.imu_angle,self.data_rate)
        self.datasets.append(new_dataset)
        # This next chunk is what we actually have to change to read from the sensors
       # self.temperature += 1
       # self.humidity += 2
       # self.location += 3
       # self.x_load += 1
       # self.y_load += 5
       # self.pot_angle += 6
       # self.imu_angle += 7

    def on_pre_leave(self):
        self.event.cancel()
        ts = TestSingleton()
        ts.clear_all()
        ts.set_height(str(config.get('height', "")))
        ts.set_plot(str(config.get('plot_num', "")))
        config.set('break_height', "N/A")
        #ts.set_pre_notes(str(config.get('height', "")))
        #ts.set_post_notes(str(config.get('height', "")))
        ts.set_operator(str(config.get('operator', "")))
        ts.set_timestamp(self.start_timestamp)
        ts.set_datasets(self.datasets)
        self.datasets = []
        self.graph.remove_plot(self.plot)
        self.graph._clear_buffer()

        #for dataset in self.datasets:

#print("Timestamp:",dataset.timestamp,"Temperature:",dataset.temperature,"Humidity:",dataset.humidity,"Location:",dataset.location,"X Load:",dataset.x_load,"Y_Load:",dataset.y_load,"Pot Angle:",dataset.pot_angle,"IMU Angle",dataset.imu_angle,"CPU Time:",dataset.cpu_time)

            #print("Timestamp:",dataset.timestamp,"Temperature:",dataset.temperature,"Data Rate:",dataset.data_rate)

```

```python
class TestSingleton:
    class __TestSingleton:
        def __init__(self):
            self.clear_all()
        def clear_all(self):
            self.height = ""
            self.plot = ""
            self.pre_notes = ""
            self.post_notes = ""
            self.operator = ""
            self.timestamp = ""
            self.datasets = []
            self.break_height = ""
    instance = None
    def __init__(self):
        if not TestSingleton.instance:
            TestSingleton.instance = TestSingleton.__TestSingleton()

    def clear_all(self):
        self.instance.clear_all()

    def print_test(self):
        print("Timestamp:", self.instance.timestamp)
        print("Height:", self.instance.height)
        print("Plot:", self.instance.plot)
        print("Operator:", self.instance.operator)
        print("Break Height:", self.instance.break_height)
        print("Pre Notes:", self.instance.pre_notes)
        print("Post Notes:", self.instance.post_notes)
        print("First Dataset Temp:", self.instance.datasets[0].temperature)
        print("Second Dataset Temp:", self.instance.datasets[1].temperature)

    def get_height(self):
        return self.instance.height
    def set_height(self, height):
        self.instance.height = height

    def get_plot(self):
        return self.instance.plot
    def set_plot(self, plot):
        self.instance.plot = plot

    def get_pre_notes(self):
        return self.instance.pre_notes
    def set_pre_notes(self, pre_notes):
        self.instance.pre_notes = pre_notes

    def get_post_notes(self):
        return self.instance.post_notes
    def set_post_notes(self, post_notes):
        self.instance.post_notes = post_notes

    def get_operator(self):
        return self.instance.operator
    def set_operator(self, operator):
        self.instance.operator = operator

    def get_timestamp(self):
        return self.instance.timestamp
    def set_timestamp(self, timestamp):
        self.instance.timestamp = timestamp

    def get_datasets(self):
        return self.instance.datasets
    def set_datasets(self, datasets):
        self.instance.datasets = datasets

    def get_break_height(self):
        return self.instance.break_height
    def set_break_height(self, break_height):
        self.instance.break_height = break_height

#class Dataset:
#    def __init__(self, name, test):
#        self.name = name
#        self.number = test

#a = TestSingleton()
#a.set_height("tall")
#d1 = Dataset("ben", "1")
#d2 = Dataset("sarah", "2")
#datasets1 = []
#datasets1.append(d1)
#datasets1.append(d2)
#a.set_datasets(datasets1)
#print("a is",a.get_height())
#print(a.get_datasets()[0].name)
#b = TestSingleton()
#b.set_height("short")
#d3 = Dataset("alexander", "3")
#d4 = Dataset("quigley", "4")
#datasets2 = []
#datasets2.append(d3)
#datasets2.append(d4)
#b.set_datasets(datasets2)
#b.clear_all()
#print("b is",b.get_height())
#print(b.get_datasets()[0].name)
#print("a is",a.get_height())
#print(a.get_datasets()[0].name)
```

```
<Test>
    canvas.before:
        Color:
            rgba: (.0, 0.9, .1, .3) if self.selected else (0, 0, 0, 1)
        Rectangle:
            pos: self.pos
            size: self.size

<TestList>
    viewclass: 'Note'
    SingleSelectableRecycleBoxLayout:

<TestsScreen>:
    name: 'tests_screen'
    GranuContainer:
        GranuSideArea:
            id: tests_buttons
        GranuContent:
            GranuTitle:
                text: 'Tests'
            BoxLayout:
                orientation: 'horizontal'
                NoteList:
                    id: tests_list
                    on_interaction:
                        root.test_buttons()
                    on_deselect_all:
                        root.default_buttons()

```

```
"""
Test in Progress
"""


from kivy.lang import Builder

from kivy.properties import ListProperty
import configurator as config



from view.BaseScreen import BaseScreen
from view.SingleSelectableList import SingleSelectableList, SingleSelectableListBehavior, SingleSelectableRecycleBoxLayout
from view.elements import *

from os import listdir
from os.path import isfile, join

from kivy.garden.graph import Graph, MeshLinePlot

Builder.load_file('view/screens/main/testing/TestsScreen.kv')

class Test(SingleSelectableListBehavior, Label):
    pass

class TestList(SingleSelectableList):
    def update(self, k, val):
        self.data = [{'text': str(x)} for x in self.list_data]


class TestsScreen(BaseScreen):

    def __init__(self, **kwargs):
        super(BaseScreen, self).__init__(**kwargs)
        self.back_button = GranuSideButton(text = 'Back')
        self.back_button.bind(on_release = self.go_back)
        self.remove_button = GranuSideButton(text = 'Remove All')
        self.remove_button.bind(on_release = self.remove_tests)
        self.export_button = GranuSideButton(text = 'Export All')
        self.export_button.bind(on_release = self.export_tests)
        self.test_details_button = GranuSideButton(text = 'Test\nDetails')
        self.test_details_button.bind(on_release = self.test_details)

    def on_pre_enter(self):
        self.test_filenames = [f for f in listdir("Tests") if isfile(join("Tests", f))]



        self.default_buttons()

        self.ids['tests_list'].list_data = self.test_filenames


    def go_back(self, obj):
        super(TestsScreen, self).back()

    def remove_tests(self, obj):
        print("We should remove all tests!")

    def export_tests(self, obj):
        print("We should export all tests!")

    def test_details(self, obj):
        print("We should show test details!")

    # Button Changes

    def default_buttons(self):
        buttons = self.ids['tests_buttons']
        buttons.clear_widgets()
        buttons.add_widget(self.back_button)
        buttons.add_widget(self.remove_button)
        buttons.add_widget(self.export_button)
        buttons.add_widget(Widget())

    def test_buttons(self):
        buttons = self.ids['tests_buttons']
        buttons.clear_widgets()
        buttons.add_widget(self.back_button)
        buttons.add_widget(self.remove_button)
        buttons.add_widget(self.export_button)
        buttons.add_widget(self.test_details_button)


    def on_leave(self):
        pass

```

```
{
    "title": "Qwerty",
    "description": "A classical US Keyboard",
    "cols": 15,
    "rows": 5,
    "normal_1": [
        ["`", "`", "`", 1],    ["1", "1", "1", 1],    ["2", "2", "2", 1],
        ["3", "3", "3", 1],    ["4", "4", "4", 1],    ["5", "5", "5", 1],
        ["6", "6", "6", 1],    ["7", "7", "7", 1],    ["8", "8", "8", 1],
        ["9", "9", "9", 1],    ["0", "0", "0", 1],    ["-", "-", "-", 1],
        ["=", "=", "=", 1],    ["\u232b", null, "backspace", 2]
    ],
    "normal_2" : [
        [" ", null, "", 1.5],  ["q", "q", "q", 1],    ["w", "w", "w", 1],
        ["e", "e", "e", 1],    ["r", "r", "r", 1],    ["t", "t", "t", 1],
        ["y", "y", "y", 1],    ["u", "u", "u", 1],    ["i", "i", "i", 1],
        ["o", "o", "o", 1],    ["p", "p", "p", 1],    ["[", "[", "[", 1],
        ["]", "]", "]", 1],    ["\\", "\\", "\\", 1.5]
    ],
    "normal_3": [
        [" ", null, "", 1.8],  ["a", "a", "a", 1],    ["s", "s", "s", 1],
        ["d", "d", "d", 1],    ["f", "f", "f", 1],    ["g", "g", "g", 1],
        ["h", "h", "h", 1],    ["j", "j", "j", 1],    ["k", "k", "k", 1],
        ["l", "l", "l", 1],    [";", ";", ";", 1],    ["'", "'", "'", 1],
        [" ", null, "", 2.2]
    ],
    "normal_4": [
        ["\u21ea", null, "capslock", 2.5],  ["z", "z", "z", 1],    ["x", "x", "x", 1],
        ["c", "c", "c", 1],    ["v", "v", "v", 1],    ["b", "b", "b", 1],
        ["n", "n", "n", 1],    ["m", "m", "m", 1],    [",", ",", ",", 1],
        [".", ".", ".", 1],    ["/", "/", "/", 1],    ["\u21ea", null, "capslock", 2.5]
    ],
    "normal_5": [
        ["space", " ", "spacebar", 13], ["\u2a2f", null, "escape", 2]
    ],
    "shift_1": [
        ["~", "~", "~", 1],    ["!", "!", "!", 1],    ["@", "@", "@", 1],
        ["#", "#", "#", 1],    ["$", "$", "$", 1],    ["%", "%", "%", 1],
        ["^", "^", "^", 1],    ["&", "&", "&", 1],    ["*", "*", "*", 1],
        ["(", "(", "(", 1],    [")", ")", ")", 1],    ["_", "_", "_", 1],
        ["+", "+", "+", 1],    ["\u232b", null, "backspace", 2]
    ],
    "shift_2": [
        [" ", null, "", 1.5],  ["Q", "Q", "Q", 1],    ["W", "W", "W", 1],
        ["E", "E", "E", 1],    ["R", "R", "R", 1],    ["T", "T", "T", 1],
        ["Y", "Y", "Y", 1],    ["U", "U", "U", 1],    ["I", "I", "I", 1],
        ["O", "O", "O", 1],    ["P", "P", "P", 1],    ["{", "{", "{", 1],
        ["}", "}", "}", 1],    ["|", "|", "|", 1.5]
    ],
    "shift_3": [
        [" ", null, "", 1.8],  ["A", "A", "A", 1],    ["S", "S", "S", 1],
        ["D", "D", "D", 1],    ["F", "F", "F", 1],    ["G", "G", "G", 1],
        ["H", "H", "H", 1],    ["J", "J", "J", 1],    ["K", "K", "K", 1],
        ["L", "L", "L", 1],    [":", ":", ":", 1],    ["\"", "\"", "\"", 1],
        [" ", null, "", 2.2]
    ],
    "shift_4": [
        ["\u21ea", null, "capslock", 2.5],  ["Z", "Z", "Z", 1],    ["X", "X", "X", 1],
        ["C", "C", "C", 1],    ["V", "V", "V", 1],    ["B", "B", "B", 1],
        ["N", "N", "N", 1],    ["M", "M", "M", 1],    ["<", "<", "<", 1],
        [">", ">", ">", 1],    ["?", "?", "?", 1.5],    ["\u21ea", null, "capslock", 2.5]
    ],
    "shift_5": [
        ["space", " ", "spacebar", 13], ["\u2a2f", null, "escape", 2]
    ]
}
```

```
<UpdateScreen>:
    name: 'update_screen'
    GranuContainer:
        GranuSideArea:
            GranuSideButton:
                text: 'Back'
                on_release:
                    root.back()
        GranuContent:
            GranuTitle:
                text: 'Update'

```

```python
from kivy.lang import Builder

from view.BaseScreen import BaseScreen

Builder.load_file('view/screens/settings/UpdateScreen.kv')

class UpdateScreen(BaseScreen):
    pass

```

```
from .connections import *

class X_Load:

    def __init__(self):
        self.load = 0.0

    def get_data(self):
        try:
            self.load = X_LOAD_CHAN.voltage*1000 #scale to milliVolts
            return self.load
        except:
            return self.load

```

```python
from .connections import *

class Y_Load:

    def __init__(self):
        self.load = 0.0

    def get_data(self):
        try:
            self.load = Y_LOAD_CHAN.voltage*1000 #scale to milliVolts
            return self.load
        except:
            return self.load

```