# CropDev GUI Documentation

*This document outlines the structure of the CropDev GUI, including the most important files. Please refer to the UML class diagram for relationships between classes. As in any case, the code is best understood through reading it:*

*https://github.com/tcl326/stalk-pusher/tree/master/gui*

## Structure Overview

The GUI uses the Pygame framework:

https://www.pygame.org/wiki/GettingStarted

The master class is CropDevMain. It instantiates the principal views, such as MainView and SettingsView. It also instantiates the Hardware and BtnInput classes. It first sets the view to MainView. The GUI layout is defined in each view class. Transitions between views happen by setting CropDevMain's view field to the given view object.

All GUI modifiable settings are in main/appData.json. All static constants are in main/defs.py.

Hardware input collection happens on a separate thread than the GUI thread.

### CropDevMain

This is the entry point of the application. The GUI evolves in the loop() function, which checks for button and keyboard input as well as updates the screen.

### Hardware Class

The hardware class constantly polls the Arduino pins for input. It parses the input according to the hardware communication protocol:

OUTDATA:
'%PCLL0101$'
'%STRM0000$'
'STOP0000$'

INDATA:
everytime Arduino gets valid command it sends out following:
'%PCLLRECE$'
'%STRMRECE$'
'%STOPRECE$'
for per call and stream, it sends:

'%[N][:[DATA_TYPE][DATA]]$'
where:
N = length of entire string
DATA_TYPE = any of 'TI', 'LO', 'TE', 'HU'
For GPS TIME:
'Time: 23:16:15.999Date: 15/12/2017'

It then delegates the input to the current view via the functions temperatureIn(), humidityIn() etc.

**BtnInput class**

The BtnInput class provides the function checkInput(), which polls all buttons using the GPIO library for changes. It then provides a method wasPressed() for each button which returns True if the button was recently pressed. It is the responsibility of the CropDevMain class to call those functions.

**View class**:

This is the superclass for each view. A view can be the Welcome screen, the Settings screen, the Testing screen etc. The view has a 4-button area on the left and a Main Area on the right. The 4-button area buttons and the Context Area are dependent on the View type. The View class has 8 input methods (btn1press, btn2press, upPress, etc.), which are triggered from the Main Window. The View manages what happens when those inputs are registered. For buttons 1-4 press, it will delegate the press to the 4-button area. For the rest, it will delegate it to the Main Area by default. The view has an __init__() in which it initializes the UI. It might have a remove UI function as well.

**Notelist**

Item that displays and manages a list of notes.

- startIndex denotes which item in the list should be displayed first
- endIndex denotes which item in the list should be displayed last

Only items between startIndex inclusive and endIndex inclusive are displayed. At the beginning, startIndex is 0; endIndex is either the number of items in list -1 or length of the array of y positions -1 (depending on the length of the list). The start and end indices are updated when the following happens:

- arrowUp AND startIndex!= 0 AND focusNum ==startIndex:
  startIndex -= 1, endIndex-=1, for i in range yNotesCents setYPos of notes between startIndex and endIndex to yNotesCents[i]
- arrowDown AND endIndex!= len(list)-1 AND focusNum ==len(yNotesCents)-1:
  startIndex += 1, endIndex+=1, for i in range yNotesCents setYPos of notes between startIndex and endIndex to yNotesCents[i]
- item from list removed (between startIndex and endIndex:

remove corresponding item from notes list, if endIndex == len(list) --> endIndex -=1,
for i in range yNotesCents setYPos of notes between startIndex and endIndex to yNotesCents[i] or that latter action can be started from the index of removal

- item added to list (between startIndex and endIndex:
  if endIndex == len(list)-1 and startIndex == 0 --> endIndex++
  add corresponding item to notes list to the end

**text.py**:

Abbreviations
s: singleline / m: multiline (which means single or multi)
p: predefined font size / d: dynamic font size
t: truncated / u: untruncated
a: automatic line switch / e: explicit line switch

Conventions
pos = [x, y] -> x: x center of bounding rect, y: y center of bounding rect
dim: = [xdim, ydim] -> xdim: bounding rect width, ydim: bounding rect height

Types of font rendering:
- spu
  *s
  *p
  *u
  *defined by x, y, font
  *used in default
- spt
  *s
  *p
  *t with ".." to fit box
  *defined by x, y, font, xdim
  *used in notelists
- mpue
  *m
  *p,
  *u
  *e
  *defined by x, y, font, ydim
  *used in Buttons
- mpta
  *m
  *p
  *t to fit box

*a
*defined by x, y, fontSize, xdim, ydim
*used in Message bodies
- mpte
*m
*p
*t with '..' to fit box
*e
*defined by x, y, fontSize, xdim, ydim
*used in Message bodies

## Update protocol

- User plugs in USB stick
- User presses UPDATE button
- GUI looks for valid update files
  -a root folder named 'update'
    -a StalkPusher folder
    -an updateData.json file
    -an updateScript.py script [OPTIONAL]
- If not found: message saying not found
- If found: message saying found and comparing versions
- If user wants to update:
  -execute Script
  -copy and paste, StalkPusher folder to desktop, overriding all but:
    -appData.json [in accordance with updateData.json]
  -update appData.json with new version number
  -set all permissions on written files and folders

  -If successful display success message and put Restart message
    -When restart, do the restart
  -If not:
    -put fail message with revert changes btn
    -if revert changes, revert changes

Special attention should be given to:
  -appData.json
    -make sure there is a default value returned when given data field does not exist yet
      -if default value returned, save it immediatly under given data field
    -make sure software can handle those default values everywhere
    -make sure able to write when given data field does not exist yet
  -test files

-make sure test file read routine is flexible enough to recognize variable start and end positions of vectors

/update
updateData.json
/cropDevice
/main
/utils
/views
/...
updateConf.json contains following info:
- new version
- replace appData.json?
- etc.

## Raspi image setup

The RasberryPi can be set up by burning the latest CropDev image onto an sd card. The following includes a tutorial on the matter:

https://www.raspberrypi.org/documentation/installation/installing-images/

Following is a list of frameworks on the latest image and the installation steps taken:

**Frameworks:**

- Matplotlib,
- Numpy
- Pygame (probably there)
- Pylab
- RPi.GPIO (probably there)

**USB handling:**

- Mount automatically
- Unmount automatically
- Set mount path
- Constantly save on USB
- https://usbmount.alioth.debian.org/
    - Automatically mounts on startup, reboot, and usb inpt
    - Constantly saves on usb

**GPIO handling:**

- http://spellfoundry.com/2016/05/29/configuring-gpio-serial-port-raspbian-jessie-including-pi-3/

**Config.txt:**

- Screen dimensions, overlay, etc.

**Rc.local:**

- Run script on startup

In the order typed, from fresh stock raspi image (April 2017 img):

////////////////////////////////////////////////////////////////////////////////////////////////////

//libs

Sudo raspi-config → enable SSH

sudo apt-get update

sudo apt-get upgrade

sudo apt-get install build-essential python-dev python-distlib python-setuptools python-pip python-wheel libzmq-dev libgdal-dev

sudo apt-get install python3-numpy

sudo apt-get install python-matplotlib

////////////////////////////////////////////////////////////////////////////////////////////////////

//gpio

sudo nano /boot/config.txt

enable_uart=1

Quit

Sudo reboot

sudo systemctl stop serial-getty@ttyS0.service

sudo systemctl disable serial-getty@ttyS0.service

sudo nano /boot/cmdline.txt

remove the line: console=serial0,115200

save and reboot

sudo nano /boot/config.txt

dtoverlay=pi3-miniuart-bt

////////////////////////////////////////////////////////////////////////////////////////////////////

//usb handling

Sudo apt-get install usbmount

Sudo nano /etc/usbmount/usbmount.conf

Prepend /home/pi/Documents/cropDevUsb to list of mounted folders

Save, reboot

File explorer>edit>preferences>disable 3rd option (show options when removable media inserted)

////////////////////////////////////////////////////////////////////////////////////////////////////

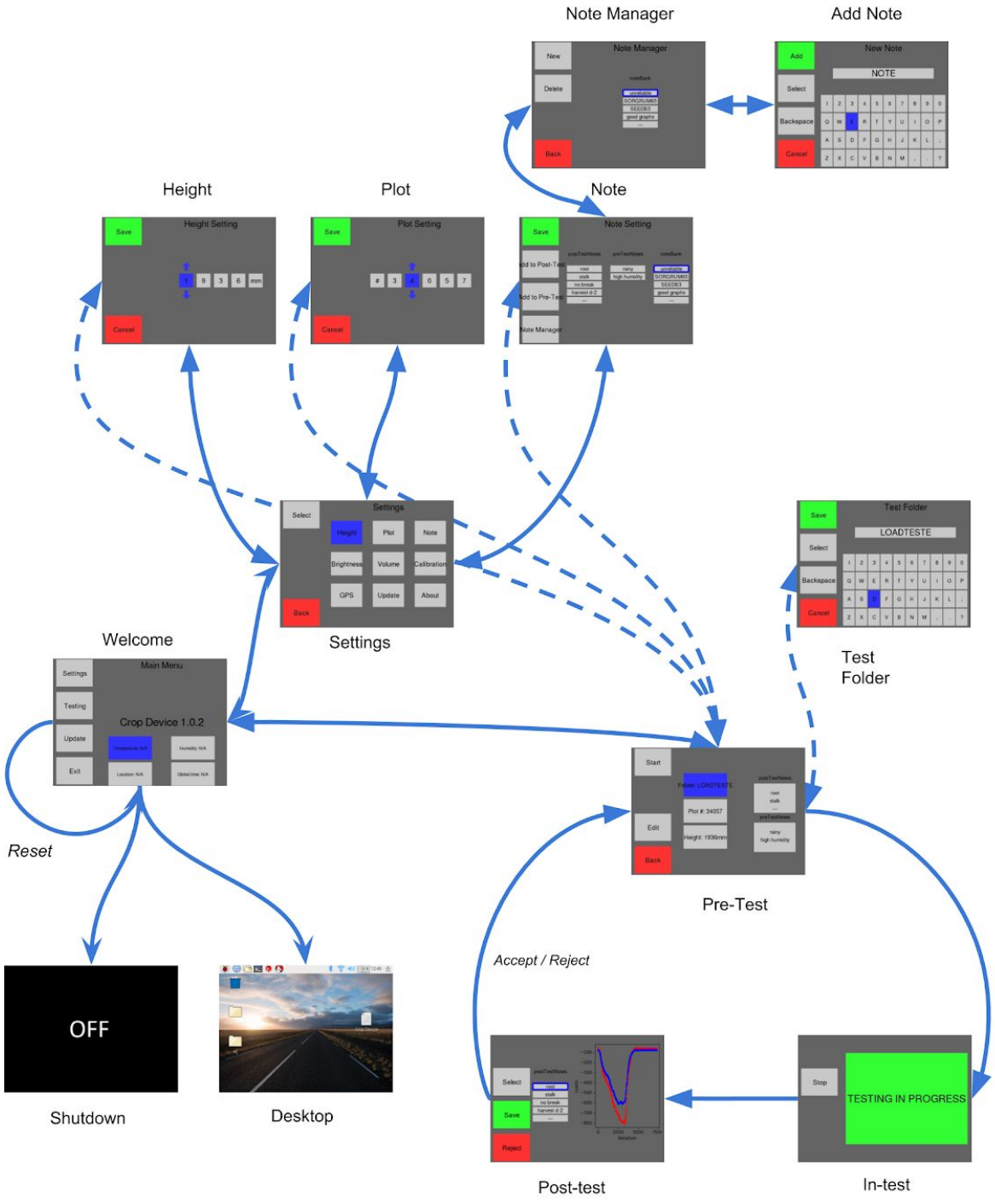
//run script on startup

Sudo nano /etc/rc.local

Add sudo python3 /home/pi/Desktop/cropDevice/main/cropDev.py before exit 0

////////////////////////////////////////////////////////////////////////////////////////////////////

**GUI Flow**


Note Manager
Add Note
Note Manager
New
Delete
Back
New Note
Add
Select
Backspace
Cancel
NOTE
Height
Plot
Note
Height Setting
Plot Setting
Note Setting
Save
Cancel
Settings
Select
Height
Plot
Note
Brightness
Volume
Calibration
GPS
Update
About
Back
Settings
Test Folder
LOADTESTE
Test
Folder
Welcome
Main Menu
Settings
Testing
Update
Exit
Crop Device 1.0.2
Reset
Start
Edit
Back
Pre-Test
Accept / Reject
OFF
Shutdown
Desktop
Select
Save
Reject
Post-test
Stop
TESTING IN PROGRESS
In-test