Monitoring and Control Software for CAEN SY127 High Voltage Supply

Will Turner, Talal Albahri, and Matt Franks

Department of Physics, University of Liverpool

E-mail: wturner@hep.ph.liv.ac.uk; talbahri@hep.ph.liv.ac.uk; mlfranks@hep.ph.liv.ac.uk

Abstract

A program with a graphical user interface (GUI) has been developed to monitor and control a CAEN SY127 High Voltage Supply, which is responsible for powering the Straw Tracker Modules for the Fermilab E989 $g - 2$ Experiment. The GUI carries out the tasks of setting and reading all the channel settings by communicating to the high voltage supply over a serial connection. This program could provide optional features such as real time plotting, trip detection and slack notifications, which enables safe, long term running. The features and the setup of the monitoring and control software are discussed in details. Moreover, a simple operating manual is included, in addition to a list of safety checks.

Introduction

The CAEN SY127 High Voltage Supply [1] was designed to supply power to a variety of detectors used in High Energy Physics Experiments. The supply is composed of several separate modules: Main Controller, Communication Controller and High Voltage Channels.

Table 1: Technical characteristics of the A333 HV module

HV Module	A333P/N
HV Full Scale	± 4/3 kV
Current Full Scale	2/3 mA
HV Resolution	1 V
Current Resolution	1 μA
•V(OVV, UNV Alarm)	50 V
$\mathbf{V_{Max}}$ Test Point Full Scale	2 V/1 kV
$\mathbf{P_{Tot}}$ Max x board	32 W
RUP, RDW Full Scale	500 V/sec
RIPPLE $\mathbf{MAX_{PP}}$ Full Load	$\leq$ 80 mV

There are two standard methods of changing settings on the CAEN HV supply. The first is to operate the supply manually using a numeric keypad and a 16 character LED display located on the front panel of the supply. All the relevant parameters of each channel may be displayed and modified by calling the appropriate ”functions”. The second method is using a built-in communication controller which provides the system with an RS232C port and a high speed serial line interface (CAENET). This in turn can be exploited by using a terminal emulator such as GTKTerm or Minicom. However, due to the limitations and time consumption of these two methods, a monitoring and control software was designed in order to allow a more sophisticated way to operate the CAEN HV supply.

The GUI is built and designed using QT Designer. This in turn can be converted to a Python source using PyQt. A main python script then utilizes the UI and PyQt source files

[1] Technical Information Manual, 1991 - `http://www.tunl.duke.edu/documents/public/electronics/CAEN/caen_sy127.pdf`

and any necessary additional libraries to act as the main GUI program.

Features

This program was written to enable quick and easy control and monitoring of the CAEN SY127 HV Supply. It allows to:

- View the status of all of the HV outputs on one overview screen.
- See a full break down, module by module of all of the outputs and their corresponding settings.
- Change any one of the channels settings from the GUI

Many optional features can be included to improve the functionality of the program such as real time plotting, trip detection and slack notifications which are utilized in Liverpool Univerity for module testing but may not necessarily be used in the experiment.

- **Real Time Plotting**: The voltages for each channel can be streamed in real time to the web service **Plotly** using the python API. This allows remote monitoring as well as easy local monitoring with an interactive live updating graph.
- **Trip Detection and Slack Notifications**: The data coming back over serial from the HV supply is monitored for the STATUS variable to change to TRIP. If this happens an automatic message is sent over **Slack** to a dedicated channel for straw module testing, this can be changed in the future to any channel wanting to be notified.

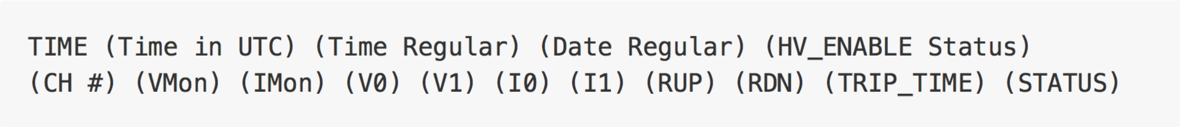
Setup

The CAEN SY127 HV supply contains a A128HS communication controller which is connected to a local machine using an RS232C serial cable. The controller also contains a permanent memory (EEPROM) which holds the current values of the parameters of all the channels in the crate. The communication settings can be set using dip switches located on the controller which allows selecting the RS232 configuration. The settings used are: Baud Rate = 4800, Number of Bits = 8, Number of Stop bits = 1 and disabled parity. The crate number is also set via these dip switches, with the crate we are controlling via serial being **crate 1** and the second supply connected via a lemo cable being **crate 2**. Then the local machine requires the following python packages: PyQt4, serial and for the extra features SLACKER (Slack API), Plotly API.

The monitoring and control software is composed of three main parts; the **HV Listener**, **HV GUI** and the **Config File**.

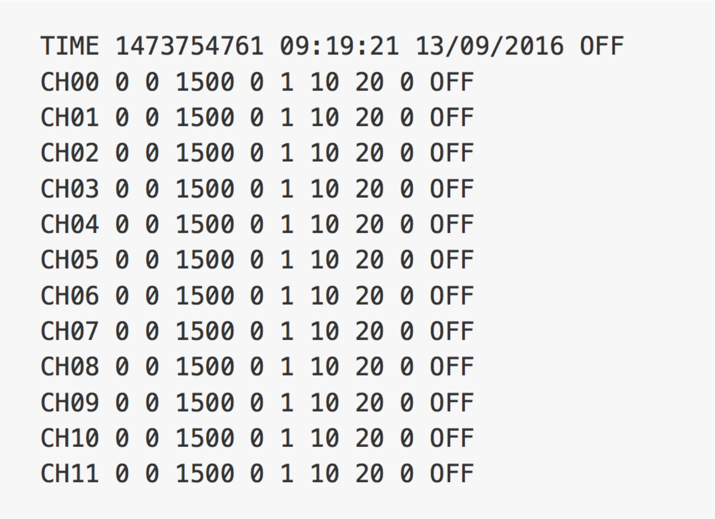
HV Listener

The HV Listener controls the HV supply to retrieve all of the data from the channels listed in the **Config File**. The HV Listener saves all information of all channels (up to 80 channels) into a **HV Data File** in the following format:



```
TIME (Time in UTC) (Time Regular) (Date Regular) (HV_ENABLE Status)
(CH #) (VMon) (IMon) (V0) (V1) (I0) (I1) (RUP) (RDN) (TRIP_TIME) (STATUS)
```

For example:



```
TIME 1473754761 09:19:21 13/09/2016 OFF
CH00 0 0 1500 0 1 10 20 0 OFF
CH01 0 0 1500 0 1 10 20 0 OFF
CH02 0 0 1500 0 1 10 20 0 OFF
CH03 0 0 1500 0 1 10 20 0 OFF
CH04 0 0 1500 0 1 10 20 0 OFF
CH05 0 0 1500 0 1 10 20 0 OFF
CH06 0 0 1500 0 1 10 20 0 OFF
CH07 0 0 1500 0 1 10 20 0 OFF
CH08 0 0 1500 0 1 10 20 0 OFF
CH09 0 0 1500 0 1 10 20 0 OFF
CH10 0 0 1500 0 1 10 20 0 OFF
CH11 0 0 1500 0 1 10 20 0 OFF
```

HV GUI

This is the part of the program the user will interface with. The HV GUI imports data from the HV Data File and displays it on the GUI. The GUI also allows users to change any channels setting on the HV supply, this is explained in more details later. To be able to send changes the HV GUI is required to communicate with the HV Listener to pause the listener from sending commands to the HV supply while the GUI sends changes. This is achieved by both files writing and reading to a file called **canRead.txt**. When the HV Listener is allowed to continue to read from the supply this file just contains **'y'**, when the HV Listner is required to stop talking to the supply the file contains **'n'** and the HV Listener confirms that it recieved the pause command by writing **'p'** to this canRead.txt file.

Config.py

This Config file is a python script which is loaded into both the HV Listener and the HV GUI and contains variables specific to the users set up. For example it contains the lists the channels from each CAEN HV supply along with the channels to each tracker module. It also displays the necessary serial information for communication with the HV module. Another important feature in the config file are the limits set for each of the channel settings. For our use the default limits are

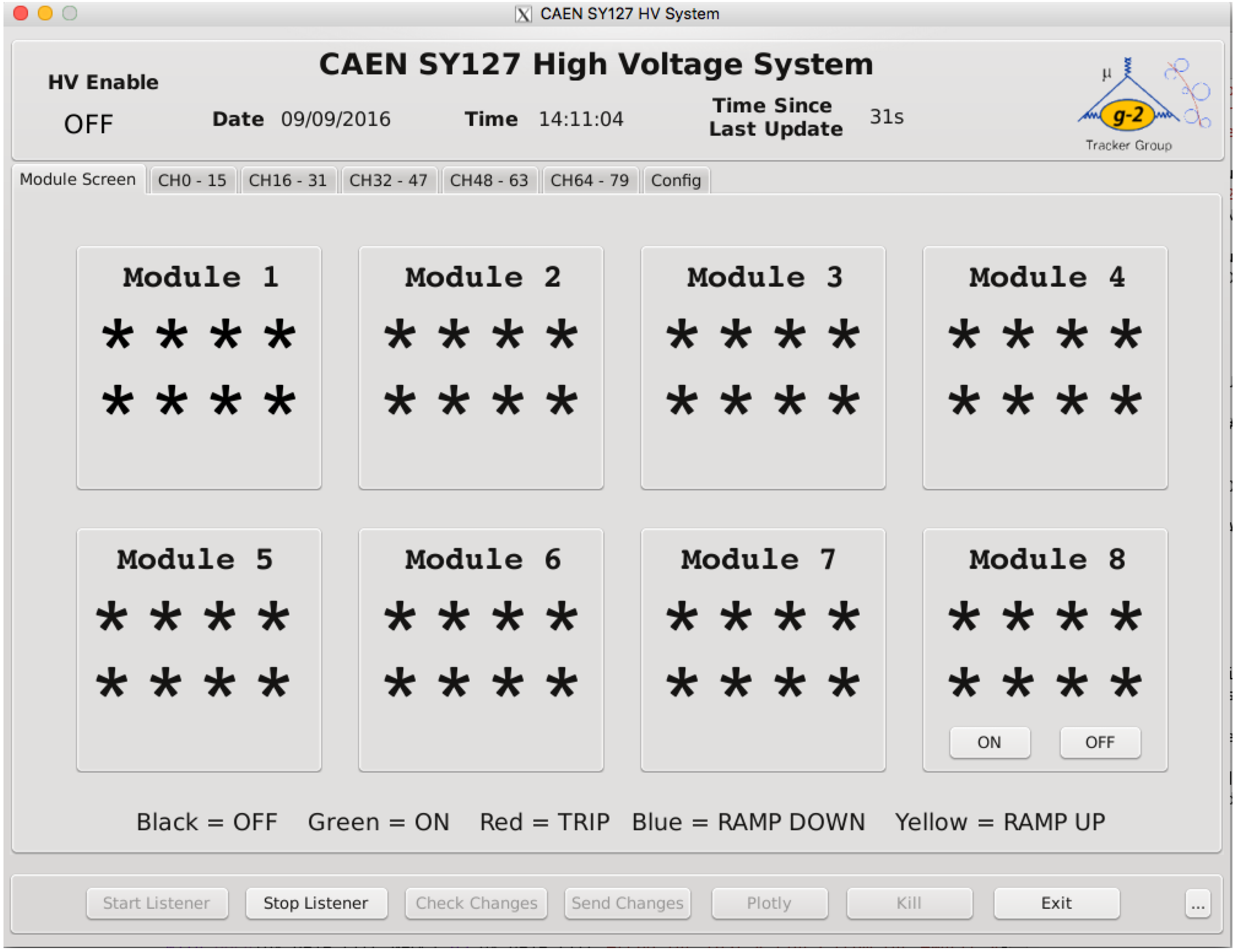
Table 2: Default limits for the channel settings

Setting	**Limit**
Max Voltage	1500V
Max Current	$10\mu A$
Ramp Up	10V/s
Ramp Down	20V/s
Trip Time	0s

This allows the user the modify the program and use the GUI according to their own setup.

How to Use

The user should first start the HV Listener either from MIDAS or running the Python script **HVListener.py** and the HV GUI by running the Python script **HVGUI.py**. Upon running the HV GUI, the user will be presented with the following GUI window.



CAEN SY127 HV System

CAEN SY127 High Voltage System

HV Enable: OFF

Date: 09/09/2016

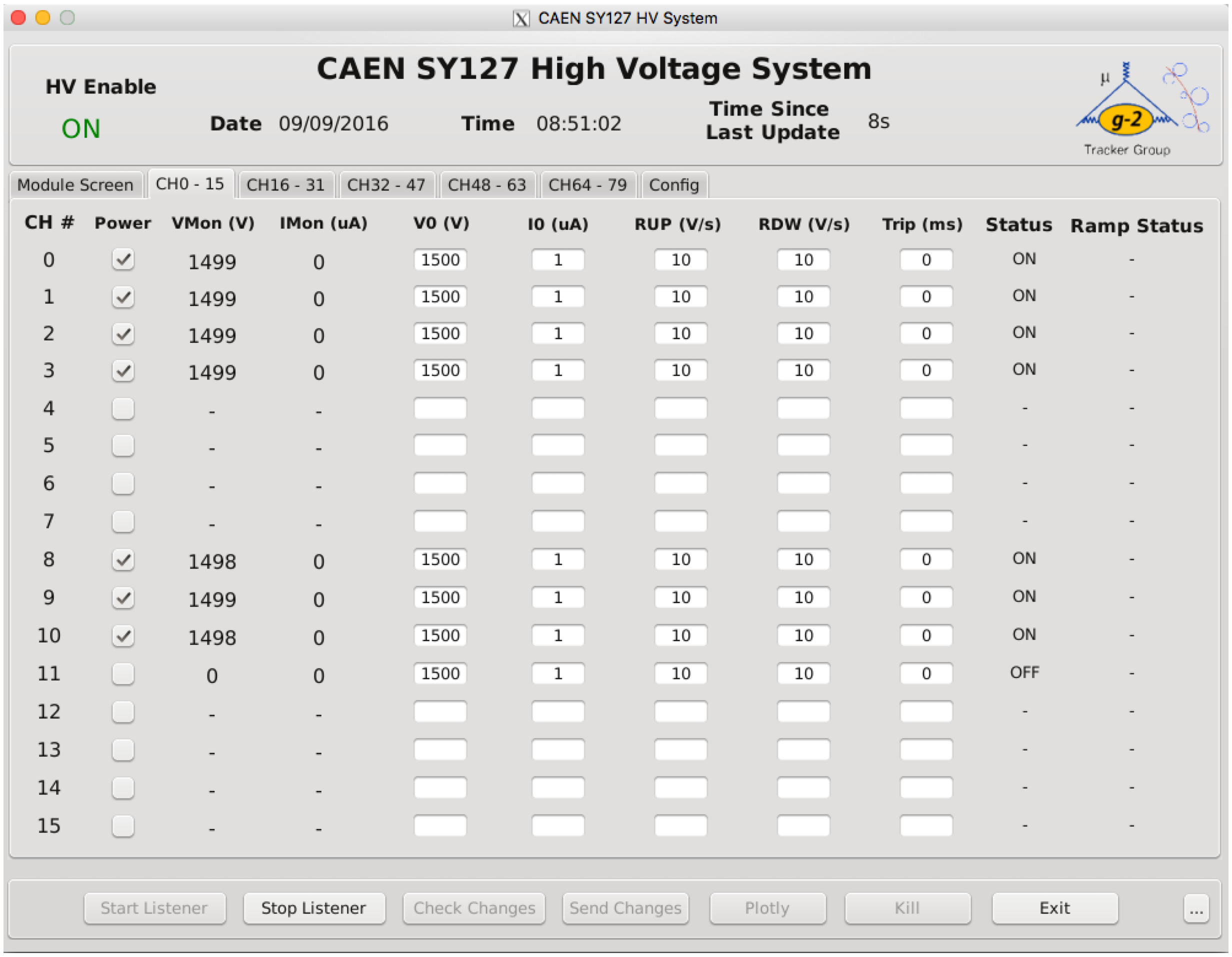
Time: 14:11:04

Time Since Last Update: 31s

Black = OFF Green = ON Red = TRIP Blue = RAMP DOWN Yellow = RAMP UP

Figure 2: The GUI for the HV system.

The main screen of the GUI shows the status of the channels of all 8 modules in a single tracker station. All the channels and setting are viewable over multiple tabs.



CAEN SY127 High Voltage System

HV Enable: ON

Date 09/09/2016

Time 08:51:02

Time Since Last Update 8s

CH #	Power	VMon (V)	IMon (uA)	V0 (V)	I0 (uA)	RUP (V/s)	RDW (V/s)	Trip (ms)	Status	Ramp Status
0	☑	1499	0	1500	1	10	10	0	ON	-
1	☑	1499	0	1500	1	10	10	0	ON	-
2	☑	1499	0	1500	1	10	10	0	ON	-
3	☑	1499	0	1500	1	10	10	0	ON	-
4	☐	-	-						-	-
5	☐	-	-						-	-
6	☐	-	-						-	-
7	☐	-	-						-	-
8	☑	1498	0	1500	1	10	10	0	ON	-
9	☑	1499	0	1500	1	10	10	0	ON	-
10	☑	1498	0	1500	1	10	10	0	ON	-
11	☐	0	0	1500	1	10	10	0	OFF	-
12	☐	-	-						-	-
13	☐	-	-						-	-
14	☐	-	-						-	-
15	☐	-	-						-	-

Figure 3: The GUI for the HV system showing a list of channels and their associated settings.

The associated parameters shown for each channel on the GUI are:

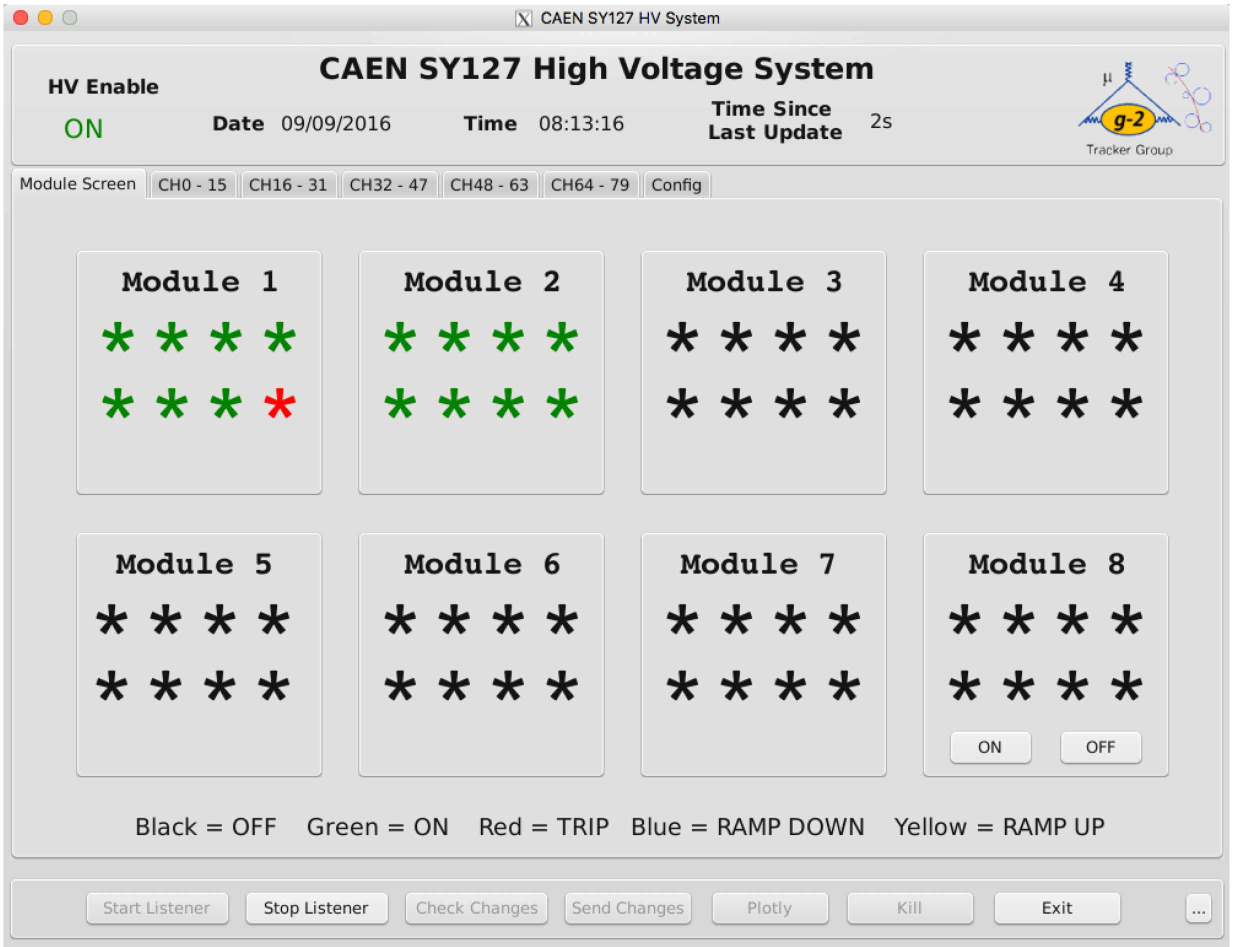
- **VMon**: The current voltage value read by the controller, expressed in V.
- **IMon**: The current current value read by the controller, expressed in μA.
- **V0**: The voltage programmed value to ramp up to, expressed in V. The maximum limit is set to 1500V for our usage.
- **I0**: The current limit programmed value, expressed in μA. The maximum limit is $10\mu A$ for our usage.
- **RUP**: The voltage ramp up speed, expressed in V/s.

- **RDW**: The voltage ramp down speed, expressed in V/s.

- **Trip**: The length of the trip time, which is the maximum time an "overcurrent" is allowed to last. If an overcurrent lasts more than the programmed value, from 1 to 9998 it will cause the channel to trip. The output voltage will drop to zero at the programmed ramp down rate and the channel will be put on the OFF state. If this parameter is set to 9999, the overcurrent may last indefinitely. If it is set to 0, the channel will be switched off as soon as an overcurrent is detected. The trip time is expressed in ms. This value is set to 0 for our usage.

- **Status**: The status, which can be ON, OFF, TRIP, OVC, OVV, UNV.
  - **ON**: The channel is On.
  - **OFF**: The channel is off.
  - **TRIP**: The channel has "tripped".
  - **OVC**: "Overcurrent", the current limit has been reached and the channel is now behaving like a constant current source.
  - **OVV**: "Overvoltage", the actual value of the high voltage output is higher than the programmed value.
  - **UNV**: "Undervoltage", the actual value of the high voltage is lower than the programmed value.

- **Ramp**: The ramp status, which can be RUP or RDW (or blank).



CAEN SY127 High Voltage System

HV Enable ON

Date 09/09/2016 Time 08:13:16 Time Since Last Update 2s

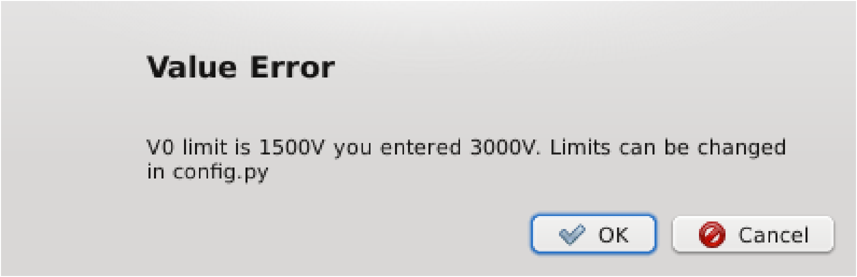
Black = OFF Green = ON Red = TRIP Blue = RAMP DOWN Yellow = RAMP UP

Figure 4: The GUI showing the status of all channels after enabling high voltage.

To modify the parameters for each channel:

- Select the tab of interest.
- Press the **Stop Listener** button to disable the monitoring.
- Turn On/Off the power of the wanted channel(s) using the tick boxes down the left hand side and change setting by using the text boxes below each setting.
- Confirm the changes by pressing the **Check Changes** button, which checks the values entered are below the allowable limit. If they are, then the settings will be sent to **Send Changes**. If the values are above the allowable limit then a warning pop-up window will be displayed and these changes will **NOT** be sent to the send changes function.

This prevents the user from entering in an incorrect value by error. IF the user did want to change a setting beyond the limits set then they would need to edit the config file manually and restart the GUI.



Value Error

V0 limit is 1500V you entered 3000V. Limits can be changed in config.py

Figure 5: Warning pop-up window

- Press **Send Changes** to forward the confirmed changes to the HV supply over the serial connection.
- Once all the changes have been made, the Listener program will resume automatically and after a few seconds the changes made should load into the GUI. The is also confirmed by the **Time Since Last Update** label.
- If the HV Listener is closed/reopened the GUI must be restarted.

Moreover, there is a debug output window which can be viewed by pressing the expansion button - see figure 3. This will expand the GUI window and display the debug window with an updated readings of the monitored parameters.

Global Changes

50x quicker than changing each channel individually. Go on Globals tab, press STOP LISTENER. Type changes (only type changes you wish to make (e.g. if current set is OK and you just want to change voltage, only change voltage global value then press change globals.)) after presssing change globals the listener thread will start up atuomatically again, and you will see your new changes when the "time since last read" reduces from what was visible previously

Safety Checks

As mentioned previously, the program checks the values of the changed settings are within the allowable range before confirming the changes. If the values are outside the allowable range then a warning pop-up window is displayed and these values will not be allowed to be sent to the supply. Moreover a **KILL** button is implemented to allow automatic disabling of the HV in case of any safety issues.

The user should also keep in mind the following safety and operating suggestions:

- Be aware that if the ”Time Since Last Update” field on the GUI displays a time larger than 2 minutes then the HV Listner script may have closed unexpectedly and the values you are seeing displayed will not update.
- Observe that the air flow within the HV supply is sufficient to prevent overheating and fires.
- Never connect any channel to any output while the HV is enabled.
- Be sure that the RS232 serial cable is properly connected to the crate.
- Never insert or remove any HV modules while is ON.
- Always make sure that the HV modules are fixed to their crates using screws.

Appendices

Appendix A: HV Listener

```python
# This script connects to the HV supply using the settings in the config file
# and saves the data from each channel to a text file. Refreshes supply values
# and loops over all channels/crates available (writen in config).

import os
import sys
import time
import serial # so we can talk over serial
from config import *

ser = serial.Serial(serial_addr, baud_rate, timeout=3)
ser.setDTR(False) #needed to keep data coming over serial without timeout
pPressed = 0 #this stores the amount of times p has been pressed on one supply.
    # each display Params window can hold 10CH values. each crate can hold 40CH's
    # so once pPressed = 4 (MAX) and if there are still channels not read from, need to change
    # crate to #2. CRATE#1 may not be full, more realistically, if CRATE#1 has 24CHs, the config
    # will have this written down

numChsTotal = len(allChs)
numCr1Chs = len(cr1Chs)
numCr2Chs = len(cr2Chs)

#####Checking CH Nums in Config######

if len(cr1Chs) == 0:
    print("No channels present for Crate#1 in config.py")
elif (len(cr1Chs) < cr1Chs[-1]+1):
    print("Length of channel list in Crate#1 is shorter than the last channel number present, you have a channel missing from the config.")
if len(cr2Chs) == 0:
    print("No channels present for Crate#2 in config.py")
elif (len(cr2Chs) < cr2Chs[-1]+1):
    print("Length of channel list in Crate#2 is shorter than the last channel number present, you have a channel missing from the config.")

print("Total Num of CHs in config " + str(numChsTotal) + " with " + str(numCr1Chs) + " in Crate#1 and " + str(numCr2Chs) + " in Crate#2")
####################################


while True:
    with open(can_Read_File, 'r', os.O_NONBLOCK) as f:
        line = f.readline()
        line = line.rstrip('\n') #logic for blocking reading while GUI is sending changes

        if line == "y": #can read
            can_Read = True
        if line == "n": #cant read and paused not acknowledged
```

```
            can_Read = False
            confirmPaused = False
        if line == "p":#cant read and paused acknowledged
            can_Read = False
            confirmPaused = True

    if can_Read == False and confirmPaused == False:
        try:
            with open(can_Read_File, 'w', os.O_NONBLOCK) as f:
                f.write("p")
        except IOError:
            print("Can not write to can_read_file")

    if can_Read == True:
        try:
            output_file = open(output_file_name, 'a', os.O_NONBLOCK) #none blocking so can write in one file and read from another
        except IOError:
            print("Can not open HV Data file to save to")

        ser.write("1".encode('ASCII')) #make sure supply is on top menu
        ser.read(9000).decode() #if you dont do something with the serial data waiting on the line it will stay there

        time.sleep(shortDelay)

        ser.write("1".encode('ASCII')) #make sure supply is on top menu
        ser.read(9000).decode() #if you dont do something with the serial data waiting on the line it will stay there

        time.sleep(shortDelay)

        ser.write("A".encode('ASCII')) #change to the display params window
        ser.read(9192).decode() #if you dont do something with the serial data waiting on the line it will stay there
    #time.sleep(shortDelay)
        ser.write("o".encode('utf-8')) #refresh params
    #put try catch around this
        serialInput = ser.read(8192).decode().strip().split('\n') # read 8192 bytes or until timeout (set to 3)

        HV_ENABLE = serialInput[22].split(" ")
        try:
            HV_ENABLE_val = HV_ENABLE[26].strip('\r')
        except IndexError:
            print("Can't find HV_ENABLE in data coming back")

        if HV_ENABLE_val not in ["ON","OFF"]:
            print("HV ENABLE STRING NOT FOUND")

        for i in range(0,26):
            serialInput.pop(0) #kills the formatting lines

        serialInput.pop()#kills the escape sequence chars (not checked this after changing pop(10) to pop() which gets last entry.

        print("HV ENABLE IS " + str(HV_ENABLE_val))

        serialDataTemp=[]
        serialDataTmp=[]
        HVData=[]

        for i in range(0,10):
            serialDataTemp.append(serialInput[i].split(' '))
            serialDataTmp.append([j for j in serialDataTemp[i] if j != ''])#strips empty ,'', from channel vars
            HVData.append([j for j in serialDataTmp[i] if j != '\r'])#strips ,'\r', from channel vars list

        for i in range(0,len(HVData)):
            print("CH" + str(i) + " is  -  " +  str(HVData[i]))

         #now here we see if the number of elements (channels) in HVData is = number of channels present.
         #if not we press 'p' to go to the next page then there should be number of chans - 9 on this page.

        print("Number of channels is " + str(numChsTotal) + ". and Lenght of HVData " + str(len(HVData)))

        while numCr1Chs > len(HVData): #if number of channels in config is > num of channels read, press p to go to next page of channels
            print("in while numCr1Chs > HVDATA -- Number of channels is " + str(numCr1Chs) + ". and Lenght of HVData " + str(len(HVData)))

         #and get the data from these ones too, save these to end of the HVData list.
            print("Num of channels is greater than the amount in Serial Data. Pressing P")
            ser.write("p".encode('utf-8')) #go to next page of channels
            serialInput_P = ser.read(8192).decode().strip().split('\n') # read 8192 bytes or until timeout (set to 3)

            for i in range(0,26):
                serialInput_P.pop(0) #kills the formatting lines

            serialInput_P.pop()#kills the escape sequence chars (not checked this after changing pop(10) to pop() which gets last entry.

            serialDataTemp_P=[]
            serialDataTmp_P=[]

            for i in range(0,len(serialInput_P)):
                serialDataTemp_P.append(serialInput_P[i].split(' '))
                serialDataTmp_P.append([j for j in serialDataTemp_P[i] if j != ''])#strips empty ,'', from channel vars
                HVData.append([j for j in serialDataTmp_P[i] if j != '\r'])#strips ,'\r', from channel vars list

            for i in range(0,len(HVData)):
                print(str(HVData[i]))
            print("end of loop")

            #done with extra channels pressing p (should have check to see if got all channels yet)

        print("number of channels is " + str(numChsTotal) + " and number of channels we have read is " + str(len(HVData)))
```

```

    del serialDataTemp[:]
    del serialDataTmp[:]

    if (len(cr2Chs) != 0):
        while numChsTotal >=  len(HVData): #if we have missing cards this may become a problem
                                            # or if we have cards present but not used and future cards in place that are used.
            print(”There are some channels in create #2 also”)
            ser.write(”1”.encode('utf-8')) #go to the main menu
            print(ser.read(9000).decode()) #if you dont do something with the serial data waiting on the line it will stay there

            ser.write(”i”.encode('utf-8')) #go to crate number menu
            ser.read(9000).decode() #if you dont do something with the serial data waiting on the line it will stay there

            ser.write(”1”.encode('utf-8')) #go to crate number 2   #CHANGE THIS TO TWO WHEN WE HAVE ANOTHER CRATE CONNECTED
            ser.read(9000).decode() #if you dont do something with the serial data waiting on the line it will stay there

            ser.write(”A”.encode('utf-8')) #display params
            ser.read(9000).decode() #if you dont do something with the serial data waiting on the line it will stay there

            serialInput = ser.read(8192).decode().strip().split('\n') # read 8192 bytes or until timeout (set to 3)

            for i in range(0,26):
                serialInput.pop(0) #kills the formatting lines

                serialInput.pop()#kills the escape sequence chars (not checked this after changing pop(10) to pop() which gets last entry.

                serialDataTemp=[]
                serialDataTmp=[]
                HVData=[]

            for i in range(0,10):
                serialDataTemp.append(serialInput[i].split(' '))
                serialDataTmp.append([j for j in serialDataTemp[i] if j != ''])#strips empty ,'', from channel vars
                HVData.append([j for j in serialDataTmp[i] if j != '\r'])#strips ,'\r', from channel vars list


    if numChsTotal <= len(HVData):
        print(”got all the channels data”)
        currentTimeOnly = time.strftime(”%H:%M:%S”)
        currentDateOnly = time.strftime(”%d/%m/%Y”)

        currentUTC      = int(time.time())
        datetime = currentTimeOnly + ” ” + currentDateOnly
        opStr = ”TIME %d %s %s” % (currentUTC,datetime,str(HV_ENABLE_val))
        output_file.write(opStr+ ”\n”)
```

```

            for i in range(0,numChsTotal):
                if HVData[i][10]=='TRIP':   #TRIP MESSAGE MAY BE DIFFERENT THAN THIS 'OVC', 'OVV' check.
                    print(HVData[i][0] + " has TRIPPED")
                    #send err to MIDAS

            for ch in HVData:
                for elem in ch:
                    try:
                        output_file.write(elem.strip())
                        output_file.write(" ")

                    except IOError:
                        print("Saving HV Data to file FAILED")

                output_file.write("\n")

        del serialDataTemp[:]
        del serialDataTmp[:]
        del HVData[:]
        del serialInput[:]


        print("done reading")
        output_file.flush()

    if can_Read == False:
        print("Cant read right now")
        time.sleep(5)

'''
        ['CH00', '0', '0', '0', '0', '1', '1', '10', '20', '0', 'OFF'] Ramp Status (optional)
HVData     [0]   , [1], [2], [3], [4], [5], [6],  [7],  [8], [9],  [10], [11]
'''
```

Appendix B: HV GUI

```python
import sys
from PyQt4 import QtCore, QtGui
import time
import HV_GUI_UI, error_GUI
import serial # so we can talk over serial
from config import *
from datetime import datetime, timedelta
from itertools import islice
import os

from dateutil.relativedelta import relativedelta
#https://pypi.python.org/pypi/python-dateutil/2.5.3

global Elements,  minheight, maxheight, height,  width, errorsignal, config, old_date
minheight = 750 #hardcoded, change UI before changing these values
maxheight = 900
height = minheight #starts with debug box hidden
width = 1000
errorsignal = 0 #errorsignal is used to pass a signal back to MainWindow from ErrorWindow to detect which button has been pressed
old_date = 0

ser = serial.Serial(serial_addr, baud_rate, timeout=3)
ser.setDTR(False)  #toggle Data Terminal Ready - makes sure all serial data is passed back

class ConnectThread(QtCore.QThread): # connect thread. so we can constantly read data
    data_downloaded = QtCore.pyqtSignal(object , object , object) #passing back three objects
                                                                  #HV_Data, time diff and HV_ENABLE status
    def __init__(self, url):
        QtCore.QThread.__init__(self)
        self.url = url
        print(url)

    def run(self):
        global RUN, old_date
        HV_DATA = []
        while (RUN == 1): # loop forever
            try:
                with open(HV_DATA_FILE_NAME) as HV_DATA_FILE:#read the last X lines from the HV_Data_file
                    tail = list(islice(reversed(list(HV_DATA_FILE)), numOfChannels+1) )   #where X is the amount of channels in the config (+1 for the time)

                for i in range(len(tail)-1,-1,-1): #puts list ordering back in correct order
                    HV_DATA.append(tail[i].strip().split("\n"))

                if old_date != HV_DATA[0]: #'checks to see if the last date entry is new. if it is send data
                    old_date = str(HV_DATA[0])
                    old_date = old_date.strip('[]').strip('\'')

                    stamp, utc_time, curr_time, date, HV_ENABLE = old_date.split()
                    #splits the top line of the data into the time, date and the HV enable string
                    #then joins the time and date back again for the difference calc.

                    old_date = str(curr_time) + " " + str(date)
                    old_date = datetime.strptime(old_date, '%H:%M:%S %d/%m/%Y')
                    current_time = datetime.now()
                    diff = relativedelta(current_time, old_date) #using the dateutil package
                    #difference between old date - new date.

                    self.data_downloaded.emit(HV_DATA, diff, HV_ENABLE) # data that is sent back.

            except IOError:
                print("Can not open HV Data file - No data sent to GUI")

            time.sleep(5) #do loop every 5sec
            del HV_DATA[:]

class TimeThread(QtCore.QThread): # time thread. so we can constantly update the time
    setTime = QtCore.pyqtSignal(object)
    print("Time Thread Started")
    def __init__(self, url):
        QtCore.QThread.__init__(self)

    def run(self):
        while True:
            #sets the labels for date and time
            currentTimeOnly = time.strftime("%H:%M:%S")
            currentDateOnly = time.strftime("%d/%m/%Y")
            datetime = currentTimeOnly + "|" + currentDateOnly
            self.setTime.emit(datetime) # data that is sent back.
            time.sleep(1)

# ==== ERROR WINDOW ==== #
class Error_Message(QtGui.QDialog, error_GUI.Ui_Dialog):
    def __init__(self, windowtitle, header, message):
        super(self.__class__, self).__init__()
        self.setupUi(self)

        self.setWindowTitle(windowtitle)
        self.Error_Title.setText(header)
        self.Error_Message.setText(message)
        self.Error_Buttons.accepted.connect(self.ok) #how to comminicate with the 'Ok' button
        self.Error_Buttons.rejected.connect(self.cancel) #how to comminicate with the 'Cancel' button

    def ok(self):
        print("Ok pressed.")

    def cancel(self):
        global errorsignal #need globalisation in function to change the global var, rather than just a local var.
        print("Cancel pressed.")
        errorsignal = 1

# ===/ERROR WINDOW==== #

# ==== Main UI Window ==== #
class HV_GUI_App(QtGui.QMainWindow, HV_GUI_UI.Ui_MainWindow):
    def __init__(self):                            # allows us to access variables, methods etc in the
    design.py file
        super(self.__class__, self).__init__()
        self.setupUi(self)                          # This is defined in HV_GUI.py file automatically

        self.threads = []
        time = TimeThread("Setting_Time")
        time.setTime.connect(self.time_set)
        self.threads.append(time)
        time.start()

    # It sets up layout and widgets that are defined

        self.setFixedSize(width, height)                # Fixes windows size. Can be resized using def
    expand below.
        self.setWindowTitle("TRACKER - CAEN SY127 HV System")

        pixmap = QtGui.QPixmap('g-2-tracker-logo-192.ico')
        self.gm2_logo.setPixmap(pixmap)#top right logo

        self.set_button.setEnabled(False)
        self.send_button.setEnabled(False)
        self.kill_button.setEnabled(False)
        self.exit_button.setEnabled(True)
        self.disconnect_button.setEnabled(False)

        self.plotly_button.setEnabled(False)

        self.connect_button.clicked.connect(self.connect)
        self.connect_button.click()#click automatically

        self.disconnect_button.clicked.connect(self.disconnect)
        self.set_button.clicked.connect(self.set)
        self.send_button.clicked.connect(self.send)
        self.kill_button.clicked.connect(self.kill)
        self.exit_button.clicked.connect(self.exit)
       # self.plotly_button.clicked.connect(self.plotlyPressed) ##removed till fixed
        self.expand_button.clicked.connect(self.expand)

        self.TextBox_btm.setReadOnly(True)

        print("GUI SET UP")

    def time_set(self, datetime):
        currentTime, currentDate = datetime.split('|')
        self.time.setText(currentTime)
        self.date.setText(currentDate)


    def connect(self): #starts the connect thread
        #this thread is used to send data back at the same time
        # as the GUI is open.
        global RUN
        RUN = 1

        try:
            with open(can_Read_File, 'w', os.O_NONBLOCK) as f: #allow listener thread to start listening
```

```
 again
                f.write("y")
                f.flush()
            print("Reading HV_DATA_FILE")

            self.TextBox_btm.moveCursor(QtGui.QTextCursor.End)
            self.TextBox_btm.insertPlainText("Retrieving data...\n")
            self.connect_button.setEnabled(False)
            self.disconnect_button.setEnabled(True)
            self.set_button.setEnabled(False)
            self.send_button.setEnabled(False)

            downloader = ConnectThread("Passing to Connect ")
            downloader.data_downloaded.connect(self.on_data_ready)
            self.threads.append(downloader)
            downloader.start()


        except IOError:
            print("Can not change can_read status in file. HV_Listner may not be running!")

    def on_data_ready(self, input_data, diff, HV_ENABLE): # shit that happens when data comes back from
 connectThread

        #might need to put a check in here to ensure data coming is in the right format.
        currentTime = time.strftime("%H:%M:%S %d/%m/%Y")

        font = QtGui.QFont("Courier")

        self.hv_enable.setText(HV_ENABLE)
        if HV_ENABLE == "ON":
            self.hv_enable.setStyleSheet('color: green')
        if HV_ENABLE == "OFF":
            self.hv_enable.setStyleSheet('color: black')


        if (diff.hours == 0) and (diff.minutes == 0):
            self.time_since_update.setText(str(diff.seconds) + "s")
        elif diff.hours == 0:
            self.time_since_update.setText(str(diff.minutes) + "min " + str(diff.seconds) + "s")
        elif diff.days == 0:
            self.time_since_update.setText(str(diff.hours) + "hr " + str(diff.minutes) + "min " + str(diff
 .seconds) + "s")
        else:
            self.time_since_update.setText(str(diff.days) + "days " + str(diff.hours) + "hrs " + str(diff.
 minutes) + "mins")

        #above sets the correct date since update in the GUI based on the time in the
        #Current Date - HV data file date. This lets you know how long its been since
        #the HV module spat data out.

        empty_channel = ["-1","-1","-1","-1","-1","-1","-1","-1","-1","-1","-1","-1"]
        #fills empty channels with spacers. these -1s are not used in the GUI but stored
        # to keep everything in correct ordering.

        while (len(input_data) != totalAmountOfOutputs+1):
            input_data.append(empty_channel)

        HV_DATA_tmp = [[] for x in range(0,79)] #make HV_DATA list of list
        HV_DATA_Arranged_list = [[] for x in range(0,79)] #make HV_DATA list of list
        HV_DATA = [[] for x in range(0,79)]
        '''   Vmon IMon
['CH00 0 0 0 0 1 1 10 20 0 OFF']
['CH01 0 0 0 0 1 1 10 20 0 OFF']
['CH02 0 0 0 0 1 1 10 20 0 OFF']
['CH03 0 0 0 0 1 1 10 20 0 OFF']
        ...

is a list

[['CH00', '0', '0', '0', '0', '1', '1', '10', '20', '0', 'OFF'], ['CH01', '0', '0', '0', '0', '1', '1', '10', '20', '0', 'OFF'], ... ,['CH11', '0', '0', '0', '0', '1', '1', '10', '20', '0', 'OFF']]

is a list of lists = HV_DATA
        '''
        # get input_data from file
        # split each elelment in each list to make list of lists
        # re-arrange elements based on channel number
        # filling blank channels with -1s

        self.TextBox_btm.setFont(font)
        global numOfChannels
        self.TextBox_btm.insertPlainText( "   VMON  IMON   V0     V1     I0    I1    RUP   RDW   T  STATUS   | " + input_data[0][0] + "\n")

        input_data.pop(0)#del date

        for i in range (0,numOfChannels):
            for j in range (0, len(input_data[i])):
                self.TextBox_btm.insertPlainText(str(input_data[i][j]))#fill text box with data from file
                self.TextBox_btm.insertPlainText("\n")

        for i in range(0, totalAmountOfOutputs):
            HV_DATA_tmp[i] = [x for y in input_data[i] for x in y.split()]

        for i in range(0,totalAmountOfOutputs):
            if HV_DATA_tmp[i][0] != "-1":
                chNum = HV_DATA_tmp[i][0].strip("CH")
                HV_DATA_Arranged_list[int(chNum)]=HV_DATA_tmp[i]

        for i in range(0,totalAmountOfOutputs):
            if (len(HV_DATA_Arranged_list[i]) == 0):
                HV_DATA[i]=empty_channel
            else:
                HV_DATA[i]=HV_DATA_Arranged_list[i]

        global glo_input_data
        glo_input_data = HV_DATA

        for i in range(0,totalAmountOfOutputs):
            if "-1" not in HV_DATA[i]:
                #FILL AND ALIGN VMON
                fill_VMon_cmd = "self.VMon_"+str(i)+".setText(HV_DATA["+str(i)+"][1])"
                eval(fill_VMon_cmd)
                VMon_Align_cmd = "self.VMon_"+str(i)+".setAlignment(QtCore.Qt.AlignCenter)"
                eval(VMon_Align_cmd)

                #FILL AND ALIGN IMON
                fill_IMon_cmd = "self.IMon_"+str(i)+".setText(HV_DATA["+str(i)+"][2])"
                eval(fill_IMon_cmd)
                IMon_Align_cmd = "self.IMon_"+str(i)+".setAlignment(QtCore.Qt.AlignCenter)"
                eval(IMon_Align_cmd)

                #FILL V0
                fill_V0 = "self.V0_"+str(i)+".setText(HV_DATA["+str(i)+"][3])"
                eval(fill_V0)
                V0_Align_cmd = "self.V0_"+str(i)+".setAlignment(QtCore.Qt.AlignCenter)"
                eval(V0_Align_cmd)

                #FILL I0
                fill_I0 = "self.I0_"+str(i)+".setText(HV_DATA["+str(i)+"][5])"
                eval(fill_I0)
                I0_Align_cmd = "self.I0_"+str(i)+".setAlignment(QtCore.Qt.AlignCenter)"
                eval(I0_Align_cmd)

                #fill RUP
                fill_RUP = "self.RUP_"+str(i)+".setText(HV_DATA["+str(i)+"][7])"
                eval(fill_RUP)
                RUP_Align_cmd = "self.RUP_"+str(i)+".setAlignment(QtCore.Qt.AlignCenter)"
                eval(RUP_Align_cmd)

                #fill Ramp Down
                fill_RDN = "self.RDN_"+str(i)+".setText(HV_DATA["+str(i)+"][8])"
                eval(fill_RDN)
                RDN_Align_cmd = "self.RDN_"+str(i)+".setAlignment(QtCore.Qt.AlignCenter)"
                eval(RDN_Align_cmd)

                #fill trip time
                fill_Trip="self.trip_"+str(i)+".setText(HV_DATA["+str(i)+"][9])"
                eval(fill_Trip)
                Trip_Align_cmd = "self.trip_"+str(i)+".setAlignment(QtCore.Qt.AlignCenter)"
                eval(Trip_Align_cmd)

                # Fills Status Info
                fill_Status ="self.S_"+str(i)+".setText(HV_DATA["+str(i)+"][10])"
                eval(fill_Status)
                Stat_Align_cmd = "self.S_"+str(i)+".setAlignment(QtCore.Qt.AlignCenter)"
                eval(Stat_Align_cmd)


                if HV_DATA[i][10] == "ON": #sets the chceck boxes for power to true
                    power_status = "self.P_"+str(i)+".setChecked(True)"
                    eval(power_status)

                elif HV_DATA[i][10] == "OFF": #sets them to false if OFF or TRIP
                    power_status = "self.P_"+str(i)+".setChecked(False)"
                    eval(power_status)

                elif HV_DATA[i][10] == "TRIP":
                    power_status = "self.P_"+str(i)+".setChecked(False)"
                    eval(power_status)

                if len(HV_DATA[i]) == 12:
                    ramp_status = "self.RS_"+str(i)+".setText(HV_DATA["+str(i)+"][11])"
                    eval(ramp_status)
                    ramp_Align_cmd = "self.RS_"+str(i)+".setAlignment(QtCore.Qt.AlignCenter)"
                    eval(ramp_Align_cmd)


                #MODULE 1 STATUS SCREEN
                if (i in module1Chs):
                    Module_Input_Data = [j for j,x in enumerate(module1Chs) if x == i] #finds the input_data number from position of channel number in module input_datas list.
                    Module_Input_Data = Module_Input_Data[0] #brings module_input_datas to correct number
                    if HV_DATA[i][10] == 'ON' and HV_ENABLE == "OFF":
                        fill_M1 = "self.M1_"+str(Module_Input_Data)+".setStyleSheet('color: orange')"
                        eval(fill_M1)
                    elif HV_DATA[i][10] == 'ON' and HV_ENABLE == "ON":
                        fill_M1 = "self.M1_"+str(Module_Input_Data)+".setStyleSheet('color: green')"
                        eval(fill_M1)
                    elif HV_DATA[i][10] == 'OFF':
                        fill_M1 = "self.M1_"+str(Module_Input_Data)+".setStyleSheet('color: black')"
                        eval(fill_M1)
                    elif HV_DATA[i][10] == 'TRIP':
                        fill_M1 = "self.M1_"+str(Module_Input_Data)+".setStyleSheet('color: red')"
                        eval(fill_M1)
                    elif  HV_DATA[i][10] == 'OVC':
                        fill_M1 = "self.M1_"+str(Module_Input_Data)+".setStyleSheet('color: pink')"
                        eval(fill_M1)


                        # put in overvoltage OVV and undervoltage UNV.

                    if len(HV_DATA[i]) == 12:
                        if (HV_DATA[i][11] == "UP"):
                            fill_M1 = "self.M1_"+str(Module_Input_Data) +".setStyleSheet('color: yellow')"
                            eval(fill_M1)
                        elif (HV_DATA[i][11] == "DOWN"):
                            fill_M1 = "self.M1_"+str(Module_Input_Data) +".setStyleSheet('color: blue')"
                            eval(fill_M1)


                #MODULE 2 STATUS SCREEN
                if (i in module2Chs):
                    Module_Input_Data = [j for j,x in enumerate(module2Chs) if x == i] #finds the input_data number from position of channel number in module input_datas list.
                    Module_Input_Data = Module_Input_Data[0]  #brings module_input_datas to correct number
                    if HV_DATA[i][10] == 'ON':
                        fill_M2 = "self.M2_"+str(Module_Input_Data)+".setStyleSheet('color: green')"
                        eval(fill_M2)
                    elif HV_DATA[i][10] == 'OFF':
                        fill_M2 = "self.M2_"+str(Module_Input_Data)+".setStyleSheet('color: black')"
                        eval(fill_M2)
                    elif HV_DATA[i][10] == 'TRIP':
```

```
                    fill_M2 = "self.M2_"+str(Module_Input_Data)+".setStyleSheet('color: red')"
                    eval(fill_M2)
                elif  HV_DATA[i][10] == 'OVC':
                    fill_M2 = "self.M2_"+str(Module_Input_Data)+".setStyleSheet('color: pink')"
                    eval(fill_M2)

                if len(HV_DATA[i]) == 12:
                    if (HV_DATA[i][11] == "UP"):
                        fill_M2 = "self.M2_"+str(Module_Input_Data)+".setStyleSheet('color: yellow')"
                        eval(fill_M1)
                    elif (HV_DATA[i][11] == "DOWN"):
                        fill_M2 = "self.M2_"+str(Module_Input_Data)+".setStyleSheet('color: blue')"
                        eval(fill_M2)


            #MODULE 3 STATUS SCREEN
            if (i in module3Chs):
                Module_Input_Data = [j for j,x in enumerate(module3Chs) if x == i] #finds the input_data number from position of channel number in module input_datas list.
                Module_Input_Data = Module_Input_Data[0]  #brings module_input_datas to correct number
                if HV_DATA[i][10] == 'ON':
                    fill_M3 = "self.M3_"+str(Module_Input_Data)+".setStyleSheet('color: green')"
                    eval(fill_M3)
                elif HV_DATA[i][10] == 'OFF':
                    fill_M3 = "self.M3_"+str(Module_Input_Data)+".setStyleSheet('color: black')"
                    eval(fill_M3)
                elif HV_DATA[i][10] == 'TRIP':
                    fill_M3 = "self.M3_"+str(Module_Input_Data)+".setStyleSheet('color: red')"
                    eval(fill_M3)
                elif  HV_DATA[i][10] == 'OVC':
                    fill_M3 = "self.M3_"+str(Module_Input_Data)+".setStyleSheet('color: pink')"
                    eval(fill_M3)

                if len(HV_DATA[i]) == 12:
                    if (HV_DATA[i][11] == "UP"):
                        fill_M3 = "self.M3_"+str(Module_Input_Data)+".setStyleSheet('color: yellow')"
                        eval(fill_M3)
                    elif (HV_DATA[i][11] == "DOWN"):
                        fill_M3 = "self.M3_"+str(Module_Input_Data)+".setStyleSheet('color: blue')"
                        eval(fill_M3)


            #MODULE 4 STATUS SCREEN
            if (i in module4Chs):
                Module_Input_Data = [j for j,x in enumerate(module4Chs) if x == i] #finds the input_data number from position of channel number in module input_datas list.
                Module_Input_Data = Module_Input_Data[0]  #brings module_input_datas to correct number
                if HV_DATA[i][10] == 'ON':
                    fill_M4 = "self.M4_"+str(Module_Input_Data)+".setStyleSheet('color: green')"
                    eval(fill_M4)
                elif HV_DATA[i][10] == 'OFF':
                    fill_M4 = "self.M4_"+str(Module_Input_Data)+".setStyleSheet('color: black')"
                    eval(fill_M4)
                elif HV_DATA[i][10] == 'TRIP':
                    fill_M4 = "self.M4_"+str(Module_Input_Data)+".setStyleSheet('color: red')"
                    eval(fill_M4)
                elif  HV_DATA[i][10] == 'OVC':
                    fill_M4 = "self.M4_"+str(Module_Input_Data)+".setStyleSheet('color: pink')"
                    eval(fill_M4)

                if len(HV_DATA[i]) == 12:
                    if (HV_DATA[i][11] == "UP"):
                        fill_M4 = "self.M4_"+str(Module_Input_Data)+".setStyleSheet('color: yellow')"
                        eval(fill_M4)
                    elif (HV_DATA[i][11] == "DOWN"):
                        fill_M4 = "self.M4_"+str(Module_Input_Data)+".setStyleSheet('color: blue')"
                        eval(fill_M4)


            #MODULE 5 STATUS SCREEN
            if (i in module5Chs):
                Module_Input_Data = [j for j,x in enumerate(module5Chs) if x == i] #finds the input_data number from position of channel number in module input_datas list.
                Module_Input_Data = Module_Input_Data[0] #brings module_input_datas to correct number
                if HV_DATA[i][10] == 'ON':
                    fill_M5 = "self.M5_"+str(Module_Input_Data)+".setStyleSheet('color: green')"
                    eval(fill_M5)
                elif HV_DATA[i][10] == 'OFF':
                    fill_M5 = "self.M5_"+str(Module_Input_Data)+".setStyleSheet('color: black')"
                    eval(fill_M5)
                elif HV_DATA[i][10] == 'TRIP':
                    fill_M5 = "self.M5_"+str(Module_Input_Data)+".setStyleSheet('color: red')"
                    eval(fill_M5)
                elif  HV_DATA[i][10] == 'OVC':
                    fill_M5 = "self.M5_"+str(Module_Input_Data)+".setStyleSheet('color: pink')"
                    eval(fill_M5)

                if len(HV_DATA[i]) == 12:
                    if (HV_DATA[i][11] == "UP"):
                        fill_M5 = "self.M5_"+str(Module_Input_Data)+".setStyleSheet('color: yellow')"
                        eval(fill_M5)
                    elif (HV_DATA[i][11] == "DOWN"):
                        fill_M5 = "self.M5_"+str(Module_Input_Data)+".setStyleSheet('color: blue')"
                        eval(fill_M5)


            #MODULE 6 STATUS SCREEN
            if (i in module6Chs):
                Module_Input_Data = [j for j,x in enumerate(module6Chs) if x == i] #finds the input_data number from position of channel number in module input_datas list.
                Module_Input_Data = Module_Input_Data[0]  #brings module_input_datas to correct number
                if HV_DATA[i][10] == 'ON':
                    fill_M6 = "self.M6_"+str(Module_Input_Data)+".setStyleSheet('color: green')"
                    eval(fill_M6)
                elif HV_DATA[i][10] == 'OFF':
                    fill_M6 = "self.M6_"+str(Module_Input_Data)+".setStyleSheet('color: black')"
                    eval(fill_M6)
                elif HV_DATA[i][10] == 'TRIP':
                    fill_M6 = "self.M6_"+str(Module_Input_Data)+".setStyleSheet('color: red')"
                    eval(fill_M6)
                elif  HV_DATA[i][10] == 'OVC':
                    fill_M6 = "self.M6_"+str(Module_Input_Data)+".setStyleSheet('color: pink')"
                    eval(fill_M6)

                if len(HV_DATA[i]) == 12:
                    if (HV_DATA[i][11] == "UP"):
                        fill_M6 = "self.M6_"+str(Module_Input_Data)+".setStyleSheet('color: yellow')"
                        eval(fill_M6)
                    elif (HV_DATA[i][11] == "DOWN"):
                        fill_M6 = "self.M6_"+str(Module_Input_Data)+".setStyleSheet('color: blue')"
                        eval(fill_M6)

            #MODULE 7 STATUS SCREEN
            if (i in module7Chs):
                Module_Input_Data = [j for j,x in enumerate(module7chs) if x == i] #finds the input_data number from position of channel number in module input_datas list.
                Module_Input_Data = Module_Input_Data[0]  #brings module_input_datas to correct number
                if HV_DATA[i][10] == 'ON':
                    fill_M7 = "self.M7_"+str(Module_Input_Data)+".setStyleSheet('color: green')"
                    eval(fill_M7)
                elif HV_DATA[i][10] == 'OFF':
                    fill_M7 = "self.M7_"+str(Module_Input_Data)+".setStyleSheet('color: black')"
                    eval(fill_M7)
                elif HV_DATA[i][10] == 'TRIP':
                    fill_M7 = "self.M7_"+str(Module_Input_Data)+".setStyleSheet('color: red')"
                    eval(fill_M7)
                elif  HV_DATA[i][10] == 'OVC':
                    fill_M7 = "self.M7_"+str(Module_Input_Data)+".setStyleSheet('color: pink')"
                    eval(fill_M7)

                if len(HV_DATA[i]) == 12:
                    if (HV_DATA[i][11] == "UP"):
                        fill_M7 = "self.M7_"+str(Module_Input_Data)+".setStyleSheet('color: yellow')"
                        eval(fill_M7)
                    elif (HV_DATA[i][11] == "DOWN"):
                        fill_M7 = "self.M7_"+str(Module_Input_Data)+".setStyleSheet('color: blue')"
                        eval(fill_M7)

            #MODULE 8 STATUS SCREEN
            if (i in module8Chs):
                Module_Input_Data = [j for j,x in enumerate(module8Chs) if x == i] #finds the input_data number from position of channel number in module input_datas list.
                Module_Input_Data = Module_Input_Data[0]  #brings module_input_datas to correct number
                if HV_DATA[i][10] == 'ON':
                    fill_M8 = "self.M8_"+str(Module_Input_Data)+".setStyleSheet('color: green')"
                    eval(fill_M8)
                elif HV_DATA[i][10] == 'OFF':
                    fill_M8 = "self.M8_"+str(Module_Input_Data)+".setStyleSheet('color: black')"
                    eval(fill_M8)
                elif HV_DATA[i][10] == 'TRIP':
                    fill_M8 = "self.M8_"+str(Module_Input_Data)+".setStyleSheet('color: red')"
                    eval(fill_M8)
                elif  HV_DATA[i][10] == 'OVC':
                    fill_M8 = "self.M8_"+str(Module_Input_Data)+".setStyleSheet('color: pink')"
                    eval(fill_M8)

                if len(HV_DATA[i]) == 12:
                    if (HV_DATA[i][11] == "UP"):
                        fill_M8 = "self.M8_"+str(Module_Input_Data)+".setStyleSheet('color: yellow')"
                        eval(fill_M8)
                    elif (HV_DATA[i][11] == "DOWN"):
                        fill_M8 = "self.M8_"+str(Module_Input_Data)+".setStyleSheet('color: blue')"
                        eval(fill_M8)


#CHECK WHAT OCMES OUT OF THE SUPPLY FOR RAMPING STATUS AND TRIP STATUS MESSAGES

def disconnect(self):
    #send signal to the listener to stop listening, so we can send changes.
    # once changes have been made make sure we change can_read.txt to y
    global RUN
    RUN = 0
    time.sleep(5)
    try:
        with open(can_Read_File, 'w', os.O_NONBLOCK) as f:
            f.write("n")
            f.flush()
            confirmPaused = "n"
            while confirmPaused == "n":
                with open(can_Read_File, 'r', os.O_NONBLOCK) as f:
                    line = f.readline()
                    confirmPaused = line.rstrip('\n') #logic for blocking reading while GUI is sending changes
                #this waits for Listener to change can_read to p to show its paused and ready for us to send changes
                print("Waiting for Paused to be confirmed")
                time.sleep(2)

        print("paused confirmed  | confirmPaused = " + str(confirmPaused))
        print("Disconnect Pressed")
        can_Read = False
        self.disconnect_button.setEnabled(False)
        self.connect_button.setEnabled(True)
        self.set_button.setEnabled(True)
        self.send_button.setEnabled(False)
    except IOError:
        print("Error in disconnect. can not save can_read file")

def set(self):
#0    1  2   3   4   5   6    7   8    9 10
#CH35 0  0   0   0   1   1   10   20   0 ON

    #for i in allChs:
    #    print("CH" + str(i) + str(glo_input_data[i]))

    global final_changes #this holds the changes AFTER they have been checked for limits
    final_changes = []

    curr_Voltage = []
    old_Voltage = []

    curr_current = []
    old_current = []

    curr_RUP = []
    old_RUP = []

    curr_RDN = []
    old_RDN = []

    curr_trip_time = []
    old_trip_time = []

    curr_power = []
    old_power = []

    for i in range (0,79):
        voltages_string = "self.V0_" + str(i) + ".text()"
        GUI_voltage = eval(voltages_string)
        curr_Voltage.append(GUI_voltage)
        old_Voltage.append(glo_input_data[i][3])

        current_string = "self.I0_" + str(i) + ".text()"
        GUI_current = eval(current_string)
        curr_current.append(GUI_current)
        old_current.append(glo_input_data[i][5])

        RUP_string = "self.RUP_" + str(i) + ".text()"
        GUI_RUP = eval(RUP_string)
        curr_RUP.append(GUI_RUP)
        old_RUP.append(glo_input_data[i][7])

        RDN_string = "self.RDN_" + str(i) + ".text()"
        GUI_RDN = eval(RDN_string)
        curr_RDN.append(GUI_RDN)
        old_RDN.append(glo_input_data[i][8])

        trip_time_string = "self.trip_" + str(i) + ".text()"
        GUI_trip_time = eval(trip_time_string)
        curr_trip_time.append(GUI_trip_time)
        old_trip_time.append(glo_input_data[i][9])

        power_string = "self.P_" + str(i) + ".checkState()"
        GUI_power = eval(power_string)
        curr_power.append(GUI_power)
        if glo_input_data[i][10] == "ON":
            old_power.append(2)
        else:
            old_power.append(0)

    print("****************")
    print("GUI Voltages")
    print(curr_Voltage)
    print("file_voltages")
    print(old_Voltage)
    print("****************")
    print("GUI currents")
    print(curr_current)
    print("file_currents")
    print(old_current)
    print(”****************”)
    print(”GUI ramp up”)
    print(curr_RUP)
    print(”file ramp up”)
    print(old_RUP)
    print(”****************”)
    print(”GUI ramp dn”)
    print(curr_RDN)
    print(”file ramp DN”)
    print(old_RDN)
    print(”****************”)
    print(”GUI trip time”)
    print(curr_trip_time)
    print(”file trip”)
    print(old_trip_time)
    print(”****************”)
    print(”GUI POWER”)
    print(curr_power)
    print(”old power”)
    print(old_power)
    print(”****************”)

    changes = [[−1 for x in range(0,7)] for y in range(0,79)]
    check_changes = [[−1 for x in range(0,7)] for y in range(0,79)]

    #compares GUI values with file values, saves each channel out, filled with −1’s for no changes.
    #need to include power functionality
    # for power, can have 0 = changed to off      1 = changed to on     −1 = no change.
    for i in allChs:
        changes[i][0] = i

        if curr_Voltage[i] != old_Voltage[i]:
            changes[i][1] = int(curr_Voltage[i])

        if curr_current[i] != old_current[i]:
            changes[i][2] = int(curr_current[i])

        if curr_RUP[i] != old_RUP[i]:
            changes[i][3] = int(curr_RUP[i])

        if curr_RDN[i] != old_RDN[i]:
            changes[i][4] = int(curr_RDN[i])

        if curr_trip_time[i] != old_trip_time[i]:
            changes[i][5] = int(curr_trip_time[i])

        if curr_power[i] != old_power[i]:
            changes[i][6] = int(curr_power[i])
        #print(”CH” + str(i) + ” is ” + str(changes[i]))
       # print(”Done”)


    #looks in each channels changes − if contains −1’s then dont send to HV changes module
    # if it contains changes send changes to HV control code.
    print(”————————————————”)
    for ch in allChs:
```

```python
            for value in range(0,7):
                if (sum(changes[ch])-int(changes[ch][0])) != -6:
                    if changes[ch][value] != -1:
                        #print("Change in CH" + str(ch))
                        if int(changes[ch][value]) > limits[value]:#Error message will pop up
                            self.dialog = Error_Message("Warning Message", "Value Error" ,"" + str(change_titles[value]) +
                                                       " limit is " + str(limits[value]) + str(units[value]) + " you entered "
                                                       + str(changes[ch][value]) + str(units[value]) + ". Limits can be changed in config.py")

                            self.dialog.exec_()
                            print("Change can not be made to " + str(change_titles[value]) + " as " + str(changes[ch][value])  +
                                  " is greater than the limit set of " + str(limits[value]) + str(units[value]) + " changing value to -1" )

                            changes[ch][value] = -1

                        else:
                            print("Make changes to " + change_titles[value] + " to " + str(changes[ch][value]) )
                            check_changes[ch][value] = int(changes[ch][value])

                            #make the send button active if no errors.
                            self.send_button.setEnabled(True)
                    else:
                        print("No changes needed for " + change_titles[value])
            print("-------------------------------")
            if (sum(changes[ch])-int(changes[ch][0])) == -6:
                print("No change needed for CH"+str(ch))

            if (sum(changes[ch])-int(changes[ch][0])) != -6:
                final_changes.append(check_changes[ch])

                #make sure the error window dosent pop up 1000's of times
                #final_changes is the array/list holding the approved changes. which is global and will be used in send.


        #print("Final changes are -")
        #print(final_changes)
#-----------------------------------------------------------------------
# change_titles = CH |    V0   |   I0   |    RUP    | RDN   |    TRIP  | POWER |
# units         =  0 |    V    |   uA   |    V/s    |  V/s  |    ms    |  NA   |
# limits        =  0 |   1500  |   1    |    10     |  20   |    0     |  0    |
# value_changes = CH |    V0   |   I0   |    RUP    | RDN   |    TRIP  | POWER |
#-----------------------------------------------------------------------
#changes to CH1, to 1500V, no change in i, change to 10V/s RUP and no other changes
#changes = (CH1,1500,-1,10,-1,-1)



    def send(self):
        print("in send func got changes " + str(final_changes))
        #in send fucntion get access to final_changes
        #these are checked values and are only changes
        for ch, chan in enumerate(final_changes):
            if ch in cr2Chs:
                print("changing to crate 2")
                #change to other crate. have to think about different ch numbers on other crate.


            time.sleep(shortDelay)

            ser.write("1".encode('utf-8'))
            time.sleep(shortDelay)

            ser.write("1".encode('utf-8'))
            time.sleep(shortDelay)

            ser.write("b".encode('utf-8'))
            time.sleep(longDelay)

            ser.write("a".encode('utf-8'))
            time.sleep(longDelay)

            print("Chaning Values of CH " + str("%02d" % final_changes[ch][0]))
            ser.write("a".encode('utf-8'))
            time.sleep(longDelay)

#            command = "CH" + "%02d" % final_changes[ch][0]
            cmd = list("CH" + "%02d" % final_changes[ch][0])

            for char in cmd:
                ser.write(str(char).encode('utf-8'))
                time.sleep(shortDelay)

            ser.write("\r\n".encode('ascii'))
            time.sleep(longDelay)

            print("Changing CH to " + str("%02d" % final_changes[ch][0]))

            if final_changes[ch][1] != -1: #voltage
                print("change voltage to " + str(final_changes[ch][1]))
                ser.write("c".encode('utf-8')) #change to voltage menu
                time.sleep(longDelay)
                cmd = list(str(final_changes[ch][1]))
                for char in cmd:
                    ser.write(str(char).encode('utf-8'))
                    time.sleep(shortDelay)
                ser.write("\r\n".encode('ascii'))
                time.sleep(shortDelay)


            if final_changes[ch][2] != -1: #current
                print("change current to " + str(final_changes[ch][2]))
                ser.write("f".encode('utf-8')) #change to current menu
                time.sleep(longDelay)
                cmd = list(str(final_changes[ch][2]))
                for char in cmd:
                    ser.write(str(char).encode('utf-8'))
                    time.sleep(shortDelay)
                ser.write("\r\n".encode('ascii'))
                time.sleep(shortDelay)


            if final_changes[ch][3] != -1: #ramp up
                print("change RUP to " + str(final_changes[ch][3]))
                ser.write("i".encode('utf-8')) #change to RUP menu
                time.sleep(longDelay)
                cmd = list(str(final_changes[ch][3]))
                for char in cmd:
                    ser.write(str(char).encode('utf-8'))
                    time.sleep(shortDelay)
                ser.write("\r\n".encode('ascii'))
                time.sleep(shortDelay)


            if final_changes[ch][4] != -1: #ramp down
                print("change RDN to " + str(final_changes[ch][4]))
                ser.write("j".encode('utf-8')) #change to RUP menu
                time.sleep(longDelay)
                cmd = list(str(final_changes[ch][4]))
                for char in cmd:
                    ser.write(str(char).encode('utf-8'))
                    time.sleep(shortDelay)
                ser.write("\r\n".encode('ascii'))
                time.sleep(shortDelay)


            if final_changes[ch][5] != -1:
                print("change trip_time to " + str(final_changes[ch][5]))
                ser.write("l".encode('utf-8')) #change to Trip Time menu
                time.sleep(longDelay)
                cmd = list(str(final_changes[ch][5]))
                for char in cmd:
                    ser.write(str(char).encode('utf-8'))
                    time.sleep(shortDelay)
                ser.write("\r\n".encode('ascii'))
                time.sleep(shortDelay)


            if final_changes[ch][6] != -1:
                print("change power to " + str(final_changes[ch][6]))
                ser.write("n".encode('utf-8')) #change to Trip Time menu
                time.sleep(longDelay)
                cmd = list(str(final_changes[ch][6]))
                for char in cmd:
                    ser.write(str(char).encode('utf-8'))
                    time.sleep(shortDelay)
                ser.write("\r\n".encode('ascii'))
                time.sleep(shortDelay)


        ser.write("1".encode('utf-8'))
        time.sleep(longDelay)
```

```python
        self.connect_button.click()#click automatically

        # start the listener thread again
        # make check button disalbed
        # make send button disabled.

    def kill(self):
        print("Kill button pressed")

        self.dialog = Settings_Window()
        self.dialog.exec_()


    def exit(self):
        print("GUI has been terminated")
        self.deleteLater()

    def expand(self):
        global minheight, maxheight, height,  width
        if height == minheight:
            self.setFixedSize(width, maxheight)
            height = maxheight
            print("Window expanded.")
        else:
            self.setFixedSize(width, minheight)
            height = minheight
            print("Window contracted.")

# ==============================================================================================

if __name__ == "__main__":
    app = QtGui.QApplication(sys.argv)
    window = HV_GUI_App()
    window.show()
    sys.exit(app.exec_())


#TODO- find a way to stop the autofil when a value is changed to be set
# plot the voltages over time (could live stream with plotly

#HV Cable # in GUI to help debug trips and stuff.

# TODO End plotly stream with  stream.close()
```

Appendix C: Config File

```

#Channel Settings (cr1 = Crate #1)
cr1Chs = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
cr2Chs = []
allChs = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
module1Chs = [0, 1, 2, 3, 4, 5, 6, 7]
module2Chs = []
module3Chs = []
module4Chs = []
module5Chs = []
module6Chs = []
module7Chs = []
module8Chs = []
unusedCHs = [8, 9, 10, 11]
numOfChannels = len(allChs)
totalAmountOfOutputs = 79


# Connection Settings
baud_rate = 4800
shortDelay = 0.55
longDelay = 0.85
serial_addr = '/dev/ttyS0'

can_Read_File = '/home/g2uol/Desktop/HV_GUI/HV_GUI_V2/can_read.txt'

# File Names
output_file_name = '/home/g2uol/Desktop/HV_GUI/HV_GUI_V2/HV_Data_3.txt'
#HV_DATA_FILE_NAME = 'HV_Data.txt'
HV_DATA_FILE_NAME = '/home/g2uol/Desktop/HV_GUI/HV_GUI_V2/HV_Data_3.txt'




#limits  Max CHs| Max V | Max I | MAX RUP | MAX RDN | Max Trip Time | Power State (unused)
# limits  =  80 |  1500 |   1   |   10    |   20    |       0       |         4
limits = [80,1500,1,10,20,0,4]
change_titles = ["CH", "V0", "I0", "RUP" , "RDN" , "TRIP_TIME", "POWER"]
units         = [ "0", "V" , "uA", "V/s" , "V/s" , "ms" , "ON/OFF"]
```