

Filter

Peter Hedlund

08/10/21

# Table of Contents

- About Me:.....3
- This Program:.....3
- Main Screen.....4
- Config Screen.....5
- .flt requirements:.....6
- .pcmd requirements.....7
- Examples.....7
  - Search log file for strings.....7
  - Generate Worksheets.....10

## About Me:

I started out as a technician, taught myself programming and design of embedded systems. I have been writing code for embedded systems since the 80's and use Python for gui applications for testing my embedded applications. I use Python as my 'crack' language when I need to get something done quickly and it helps clear my head.

## This Program:

Filter, started out as a way for me to view log files created by a compiler. While working at a company I was required to run a build file that took about 20 min to run. The output was appended to a log file.

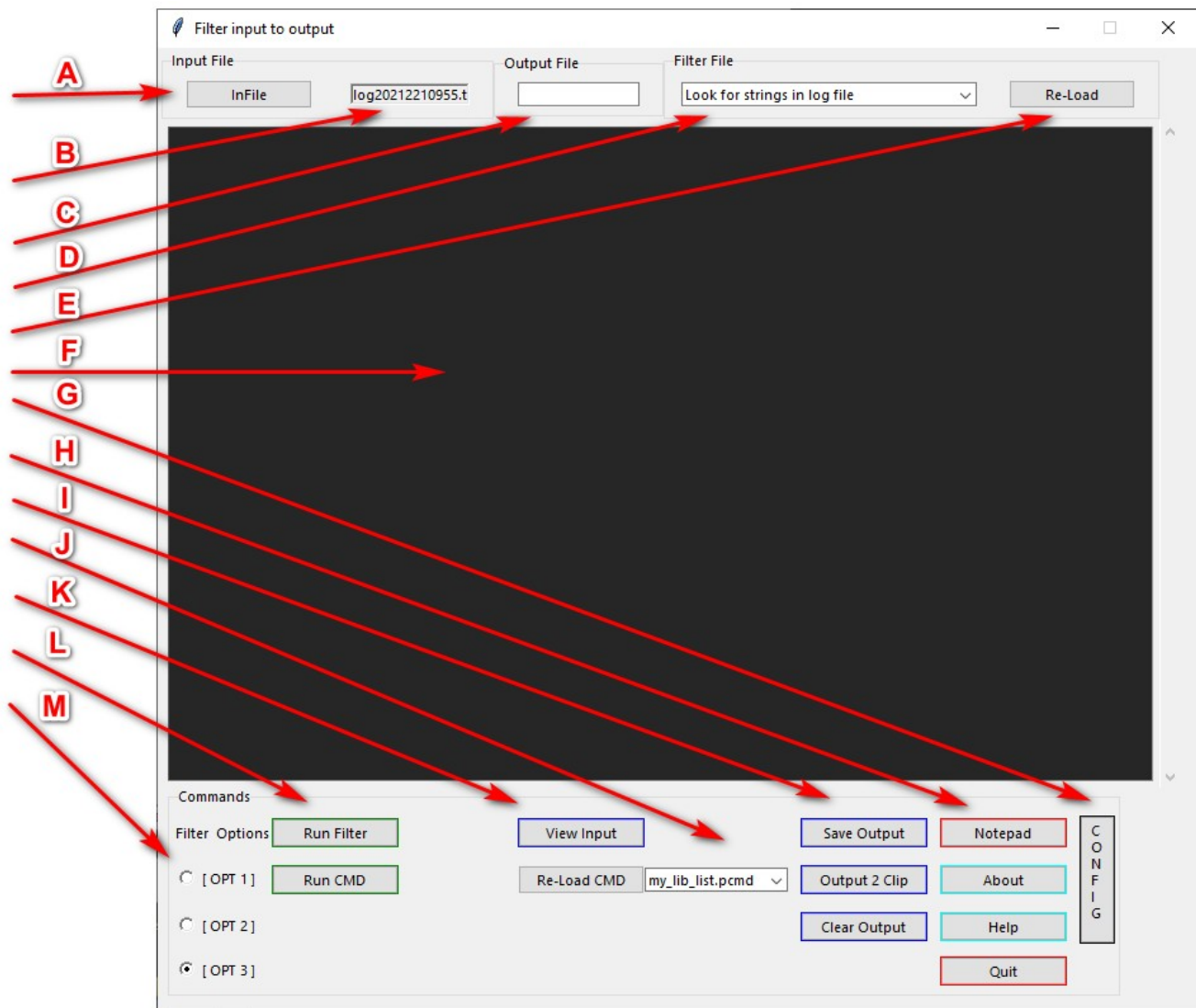
Now I needed to do two things to this log file. 1. look at it for warning and errors to fix and 2 delete the log file before running another build. Now we all know programmers like to work smarter.

This build file log was huge. I was only interested in viewing errors and warnings. I could load the log file into an editor and look for the words warning and error. That would work. But.....the engineer in me took over.

Now this situation in general was not new to me. I had many time needed to look at output file, log file, data from test outputs etc all the time. I thought what if I made a gui that could be easily customized for search tasks. The gui would not need to change, only the filter program (flt) would need to change. As far as the second part of the problem (deleting the log file) this could be done with another simple python command file (pcmd)

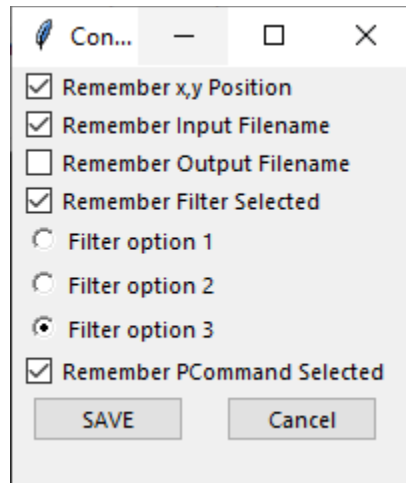
When you start up the filter program, I will load all .flt files in the flt subdirectory and all the .pcmd files in the pcmd subdirectory, and read the config file to help determine its position and to optionally remember last file selected.

# Main Screen



- A) Input File selection browse button
- B) Input file name
- C) Output File name
- D) Filter selection
- E) Reload filters
- F) Text window output area
- G) Configuration Button
- H) Generic Control buttons
- I) Text window control buttons
- J) Pcmd selection and reload button
- K) View Input file in text viewer
- L) Run Buttons for filter and Pcmd
- M) Filter option selection

## Config Screen



This screen allows the operator the option of what to save between runs of the program.

## .flt requirements:

Both .flt and .pcmd files are in python format so that you can change your editor to accept .flt and .pcmd files as python file types.

```
#-----
# Name:          template.flt
# Purpose:       Generic Template File
#               |   Maximum length of description   |
#
# Author:        USPEHED
#
# Created:
# Copyright:     (c) USPEHED 2018
# Licence:       <your licence>
#-----

# uses the following globals as input
# infile, outfile, file_save
# Functions
# scrn
#
# Sends back the following globals
# count
#

global count
count = 0

fp = open(infile, 'r')
if file_save == 1:
    fo = open(outfile, 'w')
for line in fp:
    count = count + 1

    # This is where the work is
    if output_to.get() == 1 :
        scrn.log_scrn_color("{:5d}: {}".format(count, line), 'green')
    #  PROCESS LINE HERE
    if output_to.get() == 2 :
        scrn.log_scrn_color(line, 'green')

    if file_save == 1:
        fo.write(line)
    # end of filter
if file_save == 1:
    fo.close()
```

Sample Filter file.

This file contains the code to do something with the input and output files  
If there is no filename in the output file the file\_save flag will be 0.

Output\_to.get() returns 0-2 depending on the filter options selected on the main gui (M)

Writing to the text window (F) is done using the scrn library functions  
scrn.log\_scrn('text')

scrn\_log\_scrn\_color('Text', 'color') Where color can be red, blue, green, cyan, yellow, white, violet, sky blue, hot pink, lightgrey, brown4

scrn.log\_scrn\_raw('text') does not allow color selection, meant for use with large amounts of data.

```
# Purpose:      Generic Template File
#              |      Maximum length of description      |
```

The Purpose string (see markers) is used to fill in the name of the filter. This allow more than just a filename as a description.

## .pcmd requirements

The requirements are generally the same as the .flt files with the exception of the purpose is NOT used to describe the command. The filename is.

## Examples

### Search log file for strings

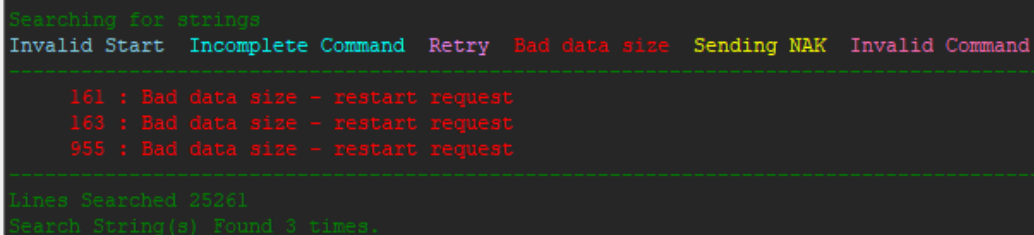
Press [InFile] and select sample/log20212220737.txt

Select the filter 'Look for strings in log file' (filename is log\_srch1.flt)

Select filter Option [OPT 3]

Press [Run Filter]

You should see the following in the text window output area (F)



```
Searching for strings
Invalid Start  Incomplete Command  Retry  Bad data size  Sending NAK  Invalid Command
-----
161 : Bad data size - restart request
163 : Bad data size - restart request
995 : Bad data size - restart request
-----
Lines Searched 25261
Search String(s) Found 3 times.
```

The log file is 25261 lines long and we are looking for 6 different strings (I used a different color for each to make viewing easier).

The filter I used was much more intricate then the sample demo as well.

It show searching each line for 6 different strings and outputting it to the screen if found.

Please note the use of my\_report for sending strings to the screen. This is just a shortcut for scrn.log\_scrn routines that can be accessed.

```
#-----
# Name:          log_srch1.flt
# Purpose:       Look for strings in log file
#               |   Maximum length of description   |
#
# Author:        USPEHED
#
# Created:
# Copyright:     (c) USPEHED 2018
# Licence:       <your licence>
#-----

# uses the following globals as input
# infile, outfile, file_save, txt
#
# Sends back the following globals
# count
#
# This filter
#
# To write to the text display of the program Use
#   my_report(_string_, _color_)
#   If color is not specified, 'green' will be used
#
#   Colors available are
#     red, yellow, green, blue, cyan, white, violet, sky blue,
#     hot pink, lightgrey, brown4

# from mylib.to_log import *

global count
global scrn
count = 0
found = 0

scn_val = ('Invalid Start', 'Incomplete Command', 'Retry', 'Bad data size',
           'Sending NAK', 'Invalid Command')

scn_col = ('sky blue', 'cyan', 'violet', 'red', 'yellow', 'hot pink')

my_report(f'\nSearching for strings \n')
for x in range(0, len(scn_val)):
    my_report(f'{scn_val[x]} ', scn_col[x])
my_report('\n')
my_report('-----')
my_report('-----\n')
fp = open(infile, 'r')
if file_save == 1:
    fo = open(outfile, 'w')
    sel = output_to.get()

for line in fp:
    count = count + 1
```



```

# This is where the work is
# None Selected
    if sel == 0:
        for y in range(0, len(sc_n_val)):
            x = line.find(sc_n_val[y])
            if x != -1:
                found += 1
            sc_n.log_sc_n_raw("{:5d}: {}".format(count, line))
            if file_save == 1:
                fo.write(line)
# Input Selected
    if sel == 1:
        show = 0
        for y in range(0, len(sc_n_val)):
            if file_save == 1:
                fo.write(line)
            x = line.find(sc_n_val[y])
            if x != -1:
                my_report(f'{count:8} : {line}', sc_n_col[y])
                found += 1
                show = 1
            if (show == 0):
                sc_n.log_sc_n_raw("{:5d}: {}".format(count, line)) # , 'green')
# Output Selected
    if sel == 2:
        for y in range(0, len(sc_n_val)):
            x = line.find(sc_n_val[y])
            if x != -1:
                my_report(f'{count:8} : {line}', sc_n_col[y])
                found += 1
                if file_save == 1:
                    fo.write(line)
        # end of filter
if file_save == 1:
    fo.close()

my_report('-----')
my_report('-----\n')
my_report(f'Lines Searched {count}\n')
my_report(f'Search String(s) Found {found} times.\n')

```

## Generate Worksheets

This Sample .pcmd will generate a worksheet for files in my library.

Select the pcmd my\_lib\_list.pcmd

Press [Run CMD]

My Python Libraries Listing			
NDX	Module Name	Comments	DONE
1	About		
2	box_char		
3	Clock_Face		
4	data_validation		
5	Env		
6	File_Log		
7	Gage		
8	Help		
9	Knob		
10	LED		
11	LED_D		
12	LED_SQ		
13	low_ascii		
14	my_math		
15	p_types		
16	print_colors		
17	Scr1_Notebook		
18	scrn_log		
19	Seven_seg		
20	Sixteen_seg		
21	to_log		
22			
23			
24			
25			
26			
27			
28			
NOTE: any and all general notes go here that you want at the bottom			

This is the .pcmd file used to generate the previous screen.

```
# from mylib.to_log import *

title = "My Python Libraries Listing"

hdr = ("NDX", "Module Name", "Comments", "DONE")

# For one page max of 28 entries
module_list = ("About", "box_char", "Clock_Face", "data_validation", "Env",
               "File_Log", "Gage", "Help", "Knob", "LED",
               "LED_D", "LED_SQ", "low_ascii", "my_math", "p_types",
               "print_colors", "Scrl_Notebook", "scrn_log",
               "Seven_seg", "Sixteen_seg", "to_log", "",
               "", "", "", "", "", "", ""
               )

need_max = 0
sort_list = 0
use_header = 1

if need_max == 1:
    mx = 0
    for x in module_list:
        if len(x) > mx:
            mx = len(x)
    my_report("Maximum Length of titles is {}\n".format(mx), 'green')
    # in this example mx was 16
else:
    a = 4
    b = 18
    c = 52
    d = 4
    # tl is the sum of all groups + (number of groups - 1)
    tl = a + b + c + d + 3
    count = 1
    ln1 = "+{}+{}+{}+{}+\n".format('=' * a, '=' * b, '=' * c, '=' * d)
    ln2 = "-{}-{}-{}-{}+\n".format('-' * a, '-' * b, '-' * c, '-' * d)
    ln3 = "+{}+\n".format('=' * tl)
    if use_header == 1:
        my_report(ln3, 'green')
        ln = tl - len(title)
        y = int(ln / 2)
        my_report("|{}{}{}|\n".format(' ' * (ln-y), title, ' ' * y), 'green')

# Header Section
my_report(ln1, 'green')
st1 = f"|{hdr[0]:>{a-1}} | {hdr[1]:^{{b-1}}}|{hdr[2]:^{{c-1}}} |{hdr[3]:{d-1}}|\n"
my_report(st1, 'green')
my_report(ln1, 'green')

# use if sort is desired
if sort_list == 1:
    module_list = sorted(module_list, key=str.lower)

# data section
for x in module_list:
    st1 = f"|{count:{{a-1}}} | {x:{{b-1}}}|{' ':{{c-1}}} |{' ':{{d}}}\n"
    my_report(st1, 'green')
```

```
        my_report(ln2, 'green')
        count = count + 1

# footer section
#   comments = ("Note: Add the following code to files after include section",
#               "#ifdef GLOBAL_SP_DEBUG",
#               "#define SERIAL_DEBUG",
#               "// #undef SERIAL_DEBUG",
#               "#endif")

comments = ("NOTE: any and all general notes go here that you want"
           " at the bottom", "")
for x in comments:
    ln = tl - 1 - len(x)
    my_report("| {}{}|\n".format(x, ' ' * ln), 'green')

my_report(ln3, 'green')
```