

File Copy Help

April 22, 2023
Thomas C. Smith

Table of Contents

Introduction	1
Main Program Form	2-3
Single-Directory Sync	3
Sub-Directory Sync	4
System Configuration	5
Log Manager	6
About	6
Source Code	7-40

Introduction:

This program is used to copy files and directories to a user-specific destination. There are two methods; 1) Single-Directory Sync and 2) Sub-Directory Sync. Each method is discussed in more details in their own section of this manual.

It may still be a work-in-progress so if there are any 'bug' found, they will be corrected in future releases.

Main Program Form:

The main program form (figure 1) is more of a summary form showing files and patterns used in each type of synchronization. The 'Single-Directory Sync' will take files and patterns for INCLUSION in the file copies, ignoring files and patterns not in the list, and the Sub-Directory will take files and patterns to EXCLUDE from recursive file copies. Recursive file copies start from the 'source' directory and copy files and subdirectories that are hyarctically below the source directory but only files and subdirectories that are not in the exclution list. At the top of the form is the 'scheduling' area where times, in 24-hour format, can be entered such that automatic file synhchronization takes place only between the 'Start' and 'Stop' times. This is activated by checking the 'Use Schedule' check box. If unchecked, no automatic file synchronization takes place. The buttons on the bootom of the form are used for manually begin synchronizing files, one for each type of sync. The form also shows the free disk information for the source and destination devices. This is useful for synchronizing files to different physical drives.

File / Directory Synchronization

File Single Subdirectory Configuration Logs Help

Schedule

Start: 0800 Stop: 1600 ☐ Use Schedule

Current Settings

Single Directory Parm's

"FITS1", "*.fits"
"FITS2", "*.fit"
"FITS3", "*.fts"

Sub Directory Parm's

"Python", "*.py"
"Text", "*.txt"
"EXEC", "*.exe"
"test2", "test2"

Disk Sizes & Paths

Single-Source Drive:	70.72G	Single-Dest Drive:	70.72G
Sub-Source Drive:	70.72G	Sub-Dest Drive:	70.72G
Single-Directory Source:	/home/pi/VirtualEnv/newCopyFile/fileCopy		
Single-Directory Destination:	/home/pi/VirtualEnv/newCopyFile/fileCopy/test1		
Sub-Directory Source:	/home/pi/VirtualEnv/newCopyFile/fileCopy		
Sub-Directory Destination:	/home/pi/VirtualEnv/newCopyFile/fileCopy/test2		

Messages

Message:

Commands

Quit Start Single Sync Start Sub Sync

Figure 1: Main Program Form

IN addition to scheduling and viewing system configuration information, there is a menu at the top of the form where the 'File', 'Single' (single-directory), 'Subdirectory' (sub-directory), 'Configuration', 'Logs', or 'Help' forms can be accessed. Each of these topics is covered later in this document.

Single-Directory Sync:

On the single-directory sync form (figure 2) there is a fields where the file extensions or full filenames can be entered and when the "Return/Enter" key is pressed the item is added to the INCLUDE list and displayed below. Below this, the second entry field is used to define the 'Target' or 'Destination' directory. This can point to a directory on the same disk as the source directory or to another 'mapped' or 'mounted' drive external to the source drive. Each entry box can be populated using the 'Browse' button next to each.

IMPORTANT: The filenames and patterns entered MUST conform to that shown and as an example, must have this format:

"FILENAME": "FILENAME" or "*.EXT": "*.EXT" such as:

"Help file": "FileCopyHelp.pdf" or "PDF files": "*.pdf"

The first part of the entry is a name that can be given for easy identification or selection in the listbox below. Technically, this entry is turned into a python dictionary entry and added to the associated dictionary variables.

An entry in the display listbox can be removed from the list by 'left-clicking' the entry line and selecting the 'Remove Item' button at the form bottom. The form color is selectable in the configuration form, discussed later on. I find it wise to have the single-directory form and the sub-directory forms a different color as they appear nearly identical and this can help avoid confusion. The 'Add Item' button adds the entry from above just like pressing 'Return/Enter' when the entry point is inside the entry field.

Sub-

Single Directory Sync

Single Directory Entries

Enter a 'Name' for the item and the 'name/extension'

File or EXT to Include: Browse

Ex: "Text": "*.txt" OR use "Browse"

Destination Directory: Browse

Count: 3

"FITS1", "*.fits"
"FITS2", "*.fit"
"FITS3", "*.fts"

Messages

Message:

Commands

Close Add Item Remove Item

Figure 2: Single-Directory Form

Directory Sync:

The 'Sub-directory form (figure 3) is similar to the 'Single-Directory' form with the additional field where directory paths can be entered such that they

are EXCLUDED from the recursive file copy. The same filename and pattern formats are used as in the 'Single-Directory' form. **THIS IS VERY IMPORTANT!**

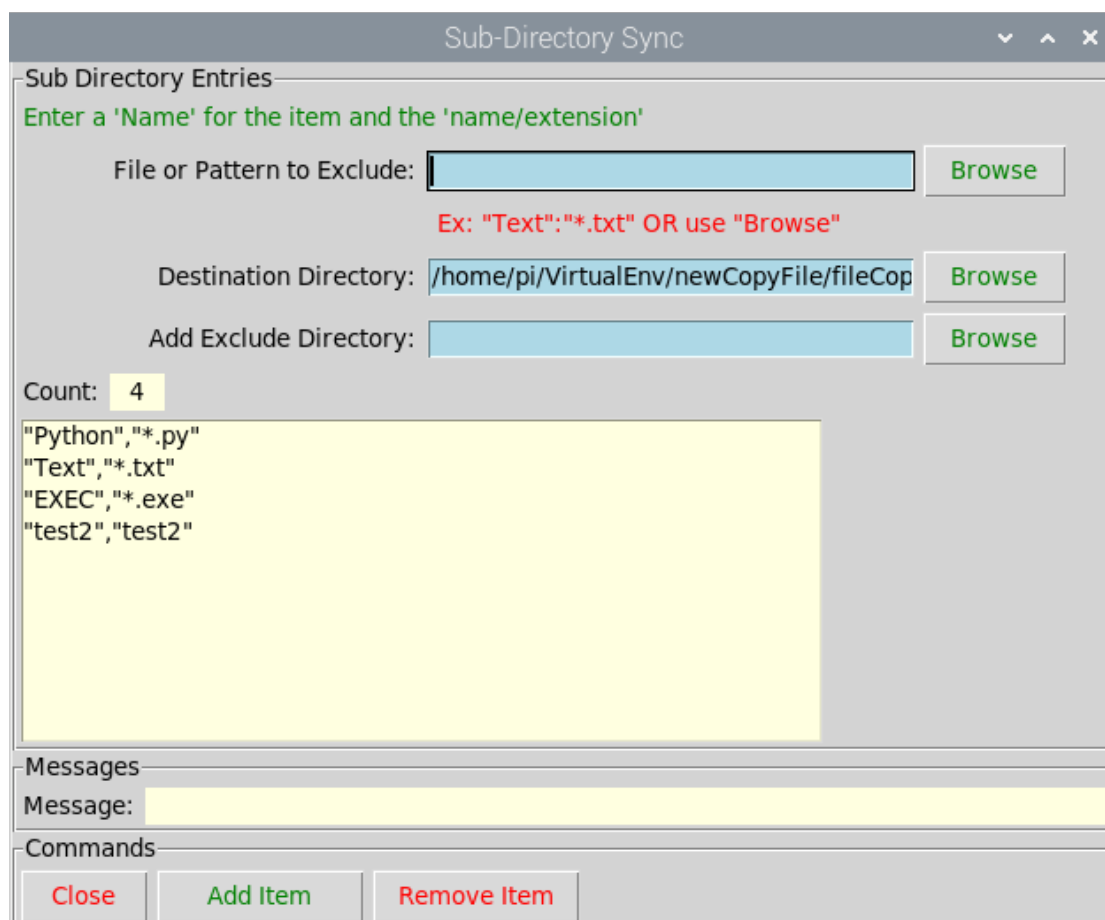
To repeat, the filenames, patterns, and directory names are those to EXCLUDE from the file copy process.

The same buttons are on the form bottom for adding an entry or removing a line from the exclusion list.

The 'Destination Directory' path is appended to the current date string so that a separate path is used for each day's file sync.

Technical: As items are added or removed from this or the Single-Directory form, a background JSON file is updated. This is the data source for the copy process.

During the manual copy process, if the destination directory path already exists you will be prompted as to overwrite the current copies. A 'No' response will simply return to the form without any copying.



Sub-Directory Sync

Sub Directory Entries

Enter a 'Name' for the item and the 'name/extension'

File or Pattern to Exclude:

Ex: "Text": "*.txt" OR use "Browse"

Destination Directory:

Add Exclude Directory:

Count: 4

"Python", "*.py"
"Text", "*.txt"
"EXEC", "*.exe"
"test2", "test2"

Messages

Message:

Commands

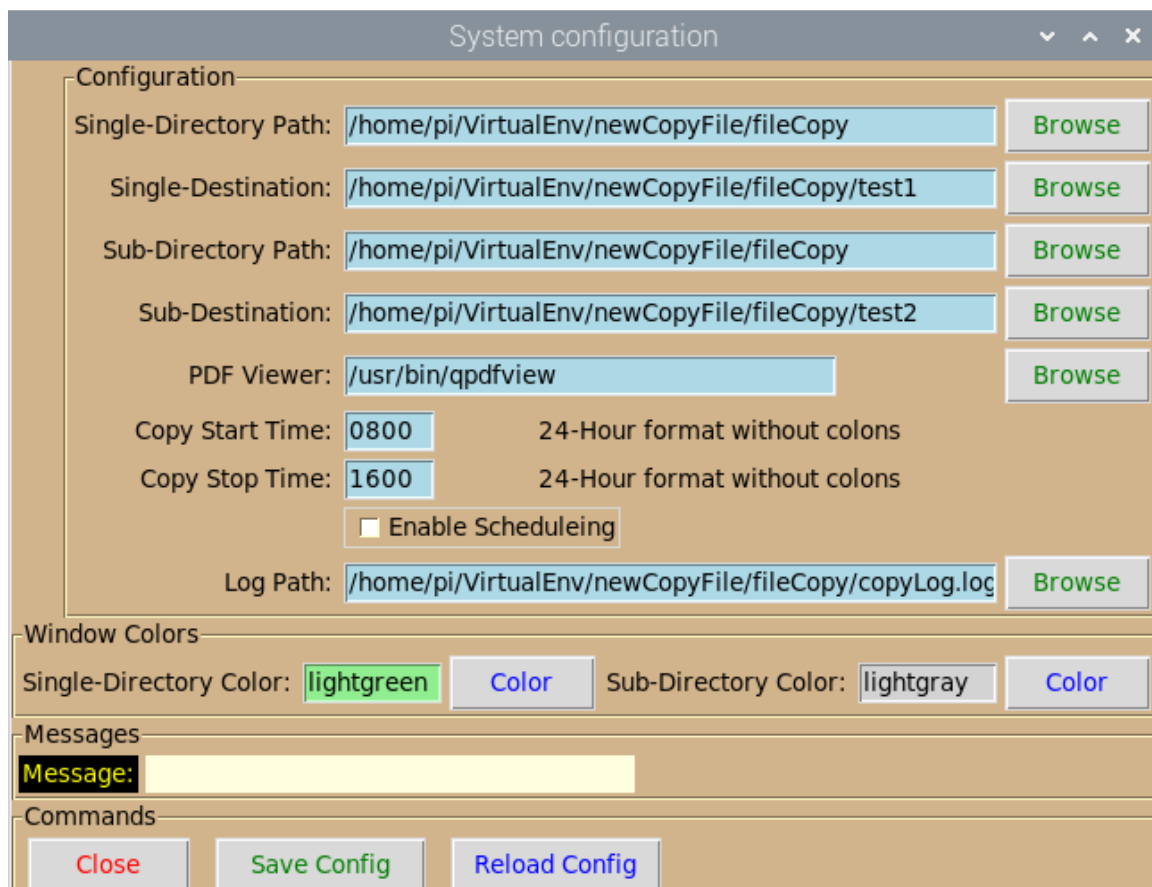
Figure 3: Sub-Directory Form

Configuration:

The 'Configuration' form is used to set the startup conditions for the program. There are separate entries for the 'Single-Directory' Source and Destination paths. These paths and the 'Copy Start Time' and 'Copy Stop Time' values are mirrored on the 'Main Program form. There is an entry field for identifying the program path used for reading this HELP PDF document. As with the other forms, the 'Browse' button can be used to navigate to the program executable file to use.

Additionally, there are two fields that dynamically reflect the colors chosen for the 'Single-Directory' form and the 'Sub-Directory' form. The color can be changed using the associated button or by manually entering a 'known' color name such as 'lightgreen'.

There is a 'Save Config' used to write out the configuration data to a JSON file, and a 'Reload Config' button that will load in the current configuration file. Once loaded affected controls and text boxes are updated.



The screenshot shows a window titled "System configuration" with a standard macOS-style title bar (minimize, maximize, close buttons). The window is divided into several sections:

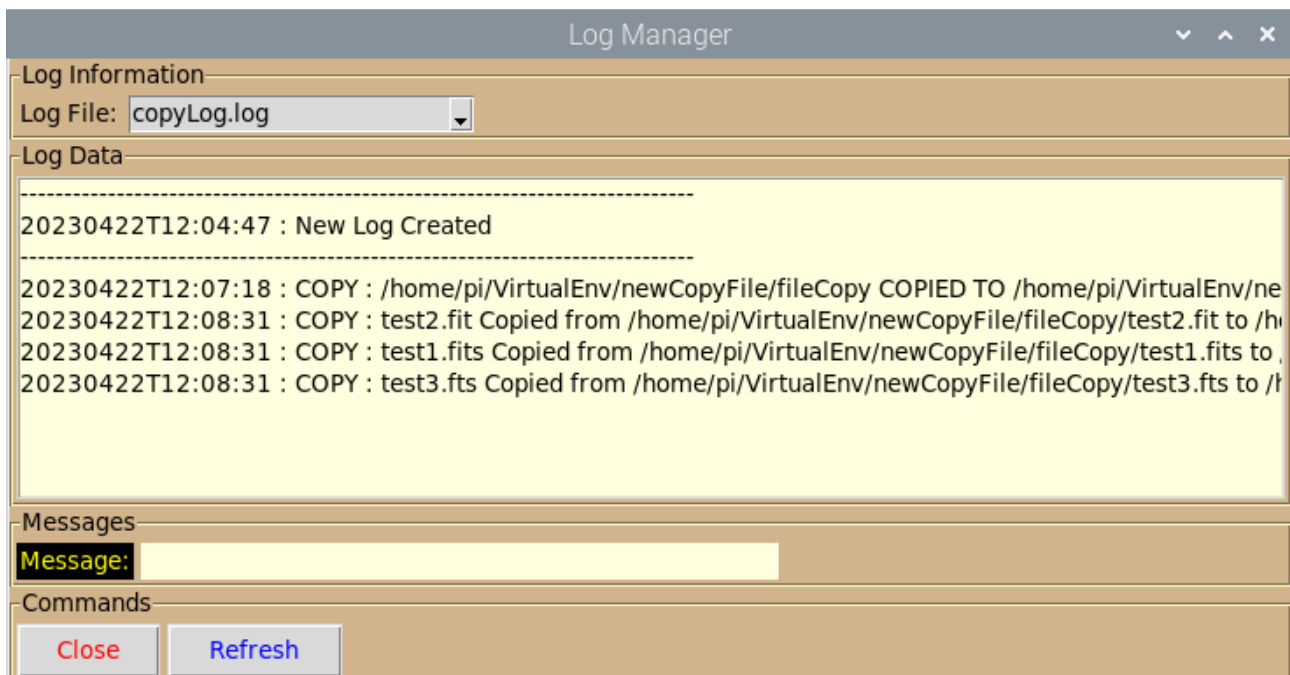
- Configuration:** This section contains several text input fields, each followed by a "Browse" button:
 - Single-Directory Path:
 - Single-Destination:
 - Sub-Directory Path:
 - Sub-Destination:
 - PDF Viewer:
 - Log Path:
- Copy Settings:** Below the PDF Viewer field, there are two spinners for "Copy Start Time" (set to 0800) and "Copy Stop Time" (set to 1600). To the right of each spinner is the text "24-Hour format without colons". Below these is a checkbox labeled "Enable Scheduling" which is currently unchecked.
- Window Colors:** This section has two color selection controls:
 - Single-Directory Color: with a "Color" button next to it.
 - Sub-Directory Color: with a "Color" button next to it.
- Messages:** A section with a label "Message:" followed by a yellow text input field.
- Commands:** A section at the bottom containing three buttons: "Close" (red text), "Save Config" (green text), and "Reload Config" (blue text).

Figure 4: System Configuration Form

Log Manager:

The 'Log Manager' form is used to display the copy activities and any errors that occur during the copy processes. The logfile is a simple TEXT FILE that can be externally copied or modified as desired.

Each time an entry is made it is time stamped and an descriptive title is included in the entry for future use in analysing the log data. Example entries are shown below in the form.



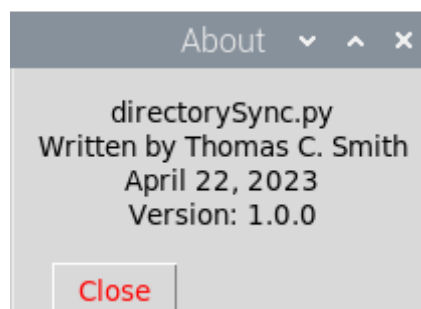
The screenshot shows a window titled "Log Manager" with a standard OS title bar (minimize, maximize, close buttons). The window is divided into several sections:

- Log Information:** Contains a dropdown menu labeled "Log File:" with "copyLog.log" selected.
- Log Data:** A large text area displaying log entries. The first entry is "20230422T12:04:47 : New Log Created". The second entry is "20230422T12:07:18 : COPY : /home/pi/VirtualEnv/newCopyFile/fileCopy COPIED TO /home/pi/VirtualEnv/ne". The third entry is "20230422T12:08:31 : COPY : test2.fit Copied from /home/pi/VirtualEnv/newCopyFile/fileCopy/test2.fit to /h". The fourth entry is "20230422T12:08:31 : COPY : test1.fits Copied from /home/pi/VirtualEnv/newCopyFile/fileCopy/test1.fits to ,". The fifth entry is "20230422T12:08:31 : COPY : test3.fts Copied from /home/pi/VirtualEnv/newCopyFile/fileCopy/test3.fts to /h".
- Messages:** A section with a label "Message:" followed by a yellow rectangular box.
- Commands:** A section containing two buttons: "Close" (red text) and "Refresh" (blue text).

Figure 5: Log Manager

About:

The 'About' form is a simple form showing the name of the program file, the author, revision date, and version number.



The screenshot shows a window titled "About" with a standard OS title bar (minimize, maximize, close buttons). The window contains the following text:

directorySync.py
Written by Thomas C. Smith
April 22, 2023
Version: 1.0.0

At the bottom of the window is a "Close" button with red text.

Source Code:

```
#!/usr/bin/python3
"""
This program is used to either automatically or manually
copy files/directories to a destination directory.

Two methods are delt with:
    Single directory copy to a single destination diretory
    Source directory and subdirectories copied to a directory
    with a specific file pattern to 'ignore'
"""

#####
#   File Name: directorySync.py                                     #
#   Written by: Thomas C. Smith                                     #
#       On: 20230411                                               #
#   Revised: 20230422                                             #
#####

#   Purpose                                                         #
#   - Provide a GUI for user interactions in creating both a      #
#       'include' json file and an 'exclude' json file of patterns. #
#   - This program can be used in manual mode or in and automatic #
#       mode based upon a scheduled start time.                   #
#   X Each mode will have its own window and saved json file.     #
#   X Need to add logging                                          #
#   X Display drive disk space on both src and dest               #
#   X The subdirectory dest path needs to have timestamp as first #
#       part of directory path and before copying if the timestamp #
#       folder already exists then use an incrementing digit as a #
#       suffix to the timestamp. User can choose to delete the dest #
#       directory using other means if they desire.               #
#####

#   Notes                                                           #
#   - Each function will have a DOCSTRING included                #
#   - <Optional> place a version on the shiBang line like:        #
#       #! /usr/var/ python3.8                                     #
#####

#   Function List                                                  #
#   - doQuit()             # Quits the program                    #
#   - setPwd()             # Gets the current working directory   #
#   - clearMessage()       # Clears the message and starts timer  #
#   - readFiles()          # Reads in the 'inclide.json' and the   #
#                           #   'exclude.json" files                #
#   - logIt()              # Logs the data to the copyLog.log'     #
#   - makeTimestamp()      # Creates a string containing a date-   #
#                           #   time format 'full' or partial are   #
#                           #   available with partial as default  #
```

```

# - openSingleDirectoryCopy() # Opens the window for working with #
#                               # the single directory configuration #
# - snglClose()               # Closes the Single Directory window #
# - clearSnglMessage()        # Clears the message and starts the #
#                               # timer #
# - snglBrowse()              # Opens the file dialog box to get user #
#                               # input for file to add to the INCLUDE#
#                               # dictionary #
# - snglDestBrowse()          # Opens the filedialog control for pick-#
#                               # ing the destination directory #
# - doAddItem(event)          # Adds the entered item into the INCLUDE#
#                               # dictionary when "Return/Enter" is #
#                               # pressed #
# - addItem()                 # Does the actual item addition #
# - removeItem()              # Removes the selected item from the #
#                               # INCLUDE dictionary #
# - writeInclude()             # Write out the 'includeDict' dictionary#
#                               # to a JSON file #
# - readInclude()              # Reads in the includeDict JSON file #
# - openSubDirectoryCopy()     # Opens the Sub-Directory copy window #
#   - subClose()               # Closes the Sub-Directory window #
#   - clearSubMessages()       # Clears the message and starts the #
#                               # timer #
#   - subBrowse()              # Opens the filedialog and allows for #
#                               # the user to select a file to add to #
#                               # the EXCLUDE dictionary #
#   - subDirBrowse()           # This opens the filedialog for the user#
#                               # to select a directory to add to the #
#                               # ECLUDE dictionary #
#   - doAddItem(event)         # Adds the entered item into the EXCLUDE#
#                               # dictionary when the user presses the#
#                               # 'Return/Enter' key #
#   - addItem()                # Does the actual item addition #
#   - removeItem()             # Removes the selected item from the #
#                               # EXCLUDE dictionary #
#   - eriteExlude()            # Write out the EXCLUDE dictionary to #
#                               # a JSON file #
#   - readExclude()            # Reads in the exclude JSON file #
# - openConfig()               # This opens the configuration window #
#   - configClose()            # Closes the configuration window #
#   - clearConfigMessage()     # Clears the message and starts the #
#                               # timer #
#   - snglBrowse()             # Opens the filedialog for the user to #
#                               # choose the source directory #
#   - snglDestBrowse()         # Opens the filedialog for the user to #
#                               # choose the destination directory #
#   - subBrowse()              # Opens the filedialog for the user to #

```



```

#                                     #   choose the source directory   #
#   - subDestBrowse()                # Opens the filedialog for the user to #
#                                     #   choose the destination directory  #
#   - saveConfig()                   # Write out the configuration data   #
#   - browsePDFViewer()              # Opens the filedialog for the user to #
#                                     #   choose the PDF Viewer program to use#
#   - snglColorChooser()             # Opens the color chooser widget for  #
#                                     #   selecting the color for the single- #
#                                     #   directory window color          #
#   -subColorChooser()               # Opens the color chooser widget for  #
#                                     #   selecting the color for the sub-  #
#                                     #   directory window color          #
#   - browseLofPath()                # Opens the filedialof for the user to #
#                                     #   choose the log file path          #
#   - openHelp()                     # Opens the 'fileCopyHelp.pdf file   #
#   - openAbout()                    # Opens the small 'about' window    #
#   - checkSchedule()                # Checks to see if we are in the time #
#                                     #   window for automatic sync action #
#   - startSingleSync()              # Starts the file sync for the single- #
#                                     #   direcotry file copying          #
#   - startSubSync()                 # Starts the file sync for the sub-  #
#                                     #   directory file copying          #
#   - doLoadConfig(event)            # Calls the loadConfig function when the#
#                                     #   user presses 'Return/Enter' key   #
#   - loadConfig()                   # Does the actual config file load   #
#   - formatSize(bytes)              # Calculates the size and units of the #
#                                     #   passed in byte count            #
#   - getDiskFreeSpace()             # Reads in the free disk space of the #
#                                     #   disk string passed in          #
#   - openLogManager()               # Opens the Log Manager window      #
#       - closeLog()                 # Closes the Log Manager window    #
#       - clearLogMessage()          # Clears the message and starts the  #
#                                     #   timer                          #
#       - getLogFiles()              # This reads in the log files        #
#       - selectLogFile(event)       # This selects the log file to use   #
#                                     #   when the user selects from list  #
#       - archiveLog()               # Not yet implemented              #
#       - refreshLog()               # Reloads the log file. This needs to #
#                                     #   be made an automatic process     #
#####

#####

#   Modules and Libraries                                     #
#####

# Comment out or delete what isn't needed

## Built-in Modules
import tkinter as tk

```

```

from tkinter.colorchooser import askcolor
from tkinter import ttk, filedialog, messagebox
import os
from os import path, listdir, mkdir, chdir, getcwd, removedirs, remove
import time
from datetime import datetime
import shutil
import subprocess
import json
# Used for disk size calculations
import sys
import io
import ctypes
import platform

## Third-party Modules

## My Modules

#####
#   End of Modules and Libraries                                     #
#####

#####
#   System Variables (Don't change unless know what you are doing) #
#####
includeDict      : dict = {} # Dictionary to hold filenames and extensions to include in copy
excludeDict      : dict = {} # Dictionary to hold filename patterns to exclude for copy
logFiles         : list = [] # List of log files
#####
#   End of System Variables                                         #
#####

#####
#   User Variables                                                 #
#####

#####
#   End of User Variables                                           #
#####

#####
#   Object Instantiations                                           #
#####

#####

```

```

# End of Object Instantiations                                     #
#####

#####

# Program Code                                                    #
#####

### Support Functions / Methods

def doQuit() -> None:
    """Quits the program."""
    sync.destroy()
# End doQuit

def setPwd() -> None:
    """This function starts us off in the
    program directory."""
    chdir("/home/pi/VirtualEnv/newCopyFile/fileCopy")
# End setPwd

def clearMessage() -> None:
    """This function clears the message and
    starts the auto-run timer."""
    message.set("")
    sync.after(15000, clearMessage)
# End clearMessage

def readFiles() -> None:
    """This function reads both the 'include.json'
    file and the 'exclude.json' file and loads
    the data into the listboxes and dictionaries
    as it appropriate."""
    global includeDict, excludeDict
    # First read in the include file
    # See if they exist first
    if (not path.exists("include.json")):
        pass

    else:
        with open("include.json", 'r') as file:
            incl = json.load(file)
            for name, value in incl.items():
                entry = '{"' + name + '":"' + value + '"'
                newentry = eval(entry)
                includeDict.update(newentry)
                MAININCLUDE.insert(tk.END, f'"{name}" , "{value}"')

    # Now read in the exclude file

```

```

# See if it exists first
if (not path.exists("exclude.json")):
    pass

else:
    with open("exclude.json", 'r') as file:
        excl = json.load(file)
        for name, value in excl.items():
            entry = '{"' + name + ':' + value + '}'
            newentry = eval(entry)
            excludeDict.update(newentry)
            MAINEXCLUDE.insert(tk.END, f'"{name}", "{value}"')

# End readFiles

def logIt(logText : str) -> None:
    """This function is used to make log entries."""
    # First see if the log exists
    if (not path.exists("copyLog.log")):
        # Nope, so create it
        logStr = "-----\n"
        logStr = logStr + f"{makeTimestamp('full')} LogFile created\n"
        logStr = logStr + "-----\n"
        with open("copyLog.log", 'w') as file:
            file.write(logStr)
            logP = getcwd() + "/copyLog.log"
            logP
            logPath.set(logP)

    with open(logPath.get(), 'a') as file:
        file.write(logText + "\n")

    message.set("Log Entry Made")

# End logIt

def makeTimestamp(part : str = "dt") -> str:
    """Creates a modified timestam for
    prepending to the destination directory."""
    # part tells this code to send back just the date part (dt) or the entire timestamp with (full)
    pre = time.localtime()
    year = str(pre.tm_year)
    month = str(f"{pre.tm_mon:02}") # This format is used to pad the 'month' so it is 2 digits with
    leading '0' if needed
    day = str(f"{pre.tm_mday:02}")
    hour = str(f"{pre.tm_hour:02}")
    minute = str(f"{pre.tm_min:02}")

```

```

second = str(f"{pre.tm_sec:02}")
if (part.lower() == "full"):
    preRet = year + month + day + "T" + hour + ":" + minute + ":" + second

elif (part.lower() == "dt"):
    preRet = year + month + day

else:
    preRet = "Invalid 'part'"
return preRet
# End makeTimestamp

```

```

def openSingleDirectoryCopy() -> None:
    """This function opens a window for controlling
    the single-directory file sync. It can be ran
    at anytime or can be constrained to times that
    are entered into the time-window fields."""
    global includeDict
    def snglClose() -> None:
        """This function closes the window."""
        sngl.destroy()
    # End snglClose

```

```

def clearSnglMessage() -> None:
    """This function clears the message and
    starts the auto-run timer."""
    snglMessage.set("")
    sngl.after(15000, clearSnglMessage)
# End clearSnglMessage

```

```

def snglBrowse() -> None:
    """This will open the filedialog for the
    selection of a filename to include."""
    # I need to add in the size of the selected drive
    snglMessage.set("Browsing")
    fName = filedialog.askopenfilename()
    if (fName != ()):
        aName = fName.split("/")
        pathLength = len(aName)
        fName = aName[pathLength - 1]
        fName = "\"" + fName + "\"\\:\"" + fName + "\""
        toInclude.set(fName)
        addItem()
    # End snglBrowse

```

```

def snglDestBrowse() -> None:
    """This will open the filedialog for the

```

```

        selection of a directory to in 'copy2'."""
        # I need to add in the size of the selected drive
        snglMessage.set("Browsing")
        fName = filedialog.askdirectory()
        if (fName != ()):
            snglDestination.set(fName)
# End snglDestBrowse

def doAddItem(event) -> None:
    """This is an event driven function."""
    addItem()
# End doAddItem

def addItem() -> None:
    """This function adds the name or extension
    to the includeDict."""
    global includeDict
    ta = "{" + toInclude.get() + "}"
    ta = eval(ta)
    includeDict.update(ta)
    INCLUDE.delete(0, tk.END)
    MAININCLUDE.delete(0, tk.END)
    for name, value in includeDict.items():
        INCLUDE.insert(tk.END, f'"{name}", "{value}"')
        MAININCLUDE.insert(tk.END, f'"{name}", "{value}"')

    snglCount.set(len(includeDict))
    writeInclude()
# End addItem

def removeItem() -> None:
    """This function removes the selected item
    from the includeDict."""
    global includeDict
    # This will be a single selected item from the 'INCLUDE'
    sel = INCLUDE.curselection()[0]          # Numerical index of the selected item
    name = INCLUDE.get(sel)                  # This is a tuple so the key is [0] and value [1]
    arrName = name.split(",")

    INCLUDE.delete(sel)                     # Removes it from the Listbox
    includeDict.pop(arrName[0][1:-1].strip()) # Removes it from the dictionary
    # Now save and reload the include.json file
    writeInclude()
    readInclude()
# End removeItem

def writeInclude() -> None:

```

```

        """This function writes out the dictionary
        'includeDict' in a json format."""
    global includeDict
    with open("include.json", 'w') as file:
        json.dump(includeDict, file)

    snglMessage.set("JSON written")
# End writeInclude

def readInclude() -> None:
    """This function reads in the 'include.json' file
    and populates the listboxes."""
    global includeDict
    INCLUDE.delete(0, tk.END)
    MAININCLUDE.delete(0, tk.END)
    # First see if it exists
    if (not path.exists("include.json")):
        # Create it
        with open("include.json", 'w') as file:
            includeDict = {}

    else:
        with open("include.json", 'r') as file:
            incl = json.load(file)
            for name, value in incl.items():
                entry = '{' + name + ':' + value + '}'
                newentry = eval(entry)
                includeDict.update(newentry)
                INCLUDE.insert(tk.END, f'"{name}"', f'"{value}"')
                MAININCLUDE.insert(tk.END, f'"{name}"', f'"{value}"')

            snglCount.set(len(includeDict))
            snglMessage.set("JSON loaded")
# End readInclude

### GUI
sngl = tk.Toplevel()
sngl.title("Single Directory Sync")
sngl.config(background = singleDirColor.get())

# Frames
snglentryframe = tk.LabelFrame(sngl, text = "Single Directory Entries", background =
singleDirColor.get())
snglentryframe.grid(row = 0, column = 0, columnspan = 5, sticky = ('e','w'))

snglmsgframe = tk.LabelFrame(sngl, text = "Messages", background = singleDirColor.get())
snglmsgframe.grid(row = 1, column = 0, columnspan = 5, sticky = ('e','w'))

```

```

snglcmdframe = tk.LabelFrame(sngl, text = "Commands", background = singleDirColor.get())
snglcmdframe.grid(row = 2, column = 0, columnspan = 5, sticky = ('e','w'))

# tkinter Variable
snglMessage      = tk.StringVar()
toInclude        = tk.StringVar()
snglCount        = tk.IntVar()

# Initializations

# Entry Area
tk.Label(snglentryframe, text = "Enter a 'Name' for the item and the 'name/extension'", background =
singleDirColor.get(), foreground = "green").grid(row = 0, column = 0, columnspan = 3, sticky = ('w'))

tk.Label(snglentryframe, text = "File or EXT to Include:", background = singleDirColor.get()).grid(row
= 1, column = 1, sticky = ('e'))
TOINCLUDE = tk.Entry(snglentryframe, textvariable = toInclude, background = "lightblue", width = 30)
TOINCLUDE.grid(row = 1, column = 2, sticky = ('w'))
SNGLBROWSE = tk.Button(snglentryframe, text = "Browse", foreground = "green", activeforeground =
"green", width = 6,
                        cursor = "hand2", command = snglBrowse)
SNGLBROWSE.grid(row = 1, column = 4, sticky = ('w'))
tk.Label(snglentryframe, text = ' Ex: "Text": "*.txt" OR use "Browse"', background =
singleDirColor.get(), foreground = "red").grid(row = 2, column = 2, columnspan = 3, sticky = ('w'))

tk.Label(snglentryframe, text = "Destination Directory:", background = singleDirColor.get()).grid(row =
3, column = 1, sticky = ('e'))
SNGLDESTINATION = tk.Entry(snglentryframe, textvariable = snglDestination, background = "lightblue",
width = 30)
SNGLDESTINATION.grid(row = 3, column = 2, sticky = ('w'))
SNGLDESTBROWSE = tk.Button(snglentryframe, text = "Browse", foreground = "green", activeforeground =
"green", width = 6,
                           cursor = "hand2", command = snglDestBrowse)
SNGLDESTBROWSE.grid(row = 3, column = 4, sticky = ('w'))

tk.Label(snglentryframe, text = "Count:", background = singleDirColor.get()).grid(row = 4, column = 0,
sticky = ('e'))
SNGLCOUNT = tk.Label(snglentryframe, textvariable = snglCount, background = "lightyellow", width = 3)
SNGLCOUNT.grid(row = 4, column = 1, sticky = ('w'))

INCLUDE = tk.Listbox(snglentryframe, background = "lightyellow", width = 40, height = 10)
INCLUDE.grid(row = 5, column = 1, columnspan = 3, sticky = ('w'))

# Message Area
tk.Label(snglmsgframe, text = "Message:", background = "black", foreground = "yellow").grid(row = 0,
column = 0, sticky = ('e'))
SNGLMESSAGE = tk.Label(snglmsgframe, textvariable = snglMessage, background = "lightyellow", width =
30)
SNGLMESSAGE.grid(row = 0, column = 1, columnspan = 3, sticky = ('w'))

```



```

# Command Area

SNGLCLOSE = tk.Button(snglcmdframe, text = "Close", foreground = "red", activeforeground = "red", width
= 5,

                        cursor = "hand2", command = snglClose)

SNGLCLOSE.grid(row = 0, column = 0, sticky = ('w'))

ADDITEM = tk.Button(snglcmdframe, text = "Add Item", foreground = "green", activeforeground = "green",
width = 10,

                        cursor = "hand2", command = addItem)

ADDITEM.grid(row = 0, column = 1, sticky = ('w'))

REMOVEITEM = tk.Button(snglcmdframe, text = "Remove Item", foreground = "red", activeforeground =
"red", width = 10,

                        cursor = "hand2", command = removeItem)

REMOVEITEM.grid(row = 0, column = 2, sticky = ('w'))

# Immediate Function Calls

clearSnglMessage()

readInclude()

TOINCLUDE.focus()

# Bindings

TOINCLUDE.bind("<Return>", doAddItem)

# Global Configurations

for child in snglentryframe.winfo_children():child.grid_configure(padx=2, pady=2)
for child in snglmsgframe.winfo_children():child.grid_configure(padx=2, pady=2)
for child in snglcmdframe.winfo_children():child.grid_configure(padx=2, pady=2)

sngl.mainloop()

# End openSingleDirectoryCopy

def openSubDirectoryCopy() -> None:
    """This function opens a window for controlling
    directory copy including sub-directories. Item
    patterns are entered into the patterns.json
    file and EXCLUDE the patterns in the file."""
    global excludeDict
    def subClose() -> None:
        """This function closes the window."""
        sub.destroy()
    # End subClose

def clearSubMessage() -> None:
    """This function clears the message and
    starts the auto-run timer."""
    subMessage.set("")

```

```

        sub.after(15000, clearSubMessage)
# End clearSubMessage

def subBrowse() -> None:
    """This will open the filedialog for the
    selection of a filename to exclude."""
    # I need to add in the size of the selected drive
    subMessage.set("Browsing")
    fName = filedialog.askopenfilename()
    if (fName != ()):
        aName = fName.split("/")
        pathLength = len(aName)
        fName = aName[pathLength - 1]
        fName = "\"" + fName + "\":" + fName + "\"\"
        toExclude.set(fName)
        addItem()
# End subBrowse

def subDirBrowse() -> None:
    """This function uses the filedialog and the
    'getdirectory' method to add the selected diretory
    to the 'excludeDict' list."""
    directory = filedialog.askdirectory()
    if (directory != ""):
        # Just get the last part of the string as we will be in the subDirectory path
        arrDir = directory.split("/")
        directory = arrDir[-1]
        toExclude.set(f'"{directory}":"{directory}"')
        excludeDirectory.set(directory)
        addItem()
# End subFolderBrowse

def subDestBrowse() -> None:
    """This will open the filedialog for the
    selection of a directory to use in 'copytree'."""
    # I need to add in the size of the selected drive
    subMessage.set("Browsing")
    fName = filedialog.askdirectory()
    if (fName != ()):
        fName = fName + "_" + makeTimestamp()
        subDestination.set(fName)
# End subDestBrowse

def doAddItem(event) -> None:
    """This is an event driven function."""
    addItem()

```

```

# End doAddItem

def addItem() -> None:
    """This function adds the name or extension
    to the includeDict."""
    global excludeDict
    ta = "{" + toExclude.get() + "}"
    ta = eval(ta)
    excludeDict.update(ta)
    EXCLUDE.delete(0, tk.END)
    MAINEXCLUDE.delete(0, tk.END)
    for name, value in excludeDict.items():
        EXCLUDE.insert(tk.END, f'"{name}", "{value}"')
        MAINEXCLUDE.insert(tk.END, f'"{name}", "{value}"')

    subCount.set(len(excludeDict))
    writeExclude()
# End addItem

def removeItem() -> None:
    """This function removes the selected item
    from the includeDict."""
    global excludeDict
    # This will be a single selected item from the 'EXCLUDE'
    sel = EXCLUDE.curselection()[0]          # Numerical index of the selected item
    name = EXCLUDE.get(sel)                  # This is a tuple so the key is [0] and value [1]
    arrName = name.split(",")

    EXCLUDE.delete(sel)                      # Removes it from the Listbox
    excludeDict.pop(arrName[0][1:-1].strip()) # Removes it from the dictionary
    # Now save and reload the exclude.json file
    writeExclude()
    readExclude()
# End removeItem

def writeExclude() -> None:
    """This function writes out the dictionary
    'excludeDict' in a json format."""
    global excludeDict
    with open("exclude.json", 'w') as file:
        json.dump(excludeDict, file)

    subMessage.set("JSON written")
# End writeExclude

def readExclude() -> None:
    """This function reads in the 'exclude.json' file

```

```

and populates the listboxes."""
global excludeDict
EXCLUDE.delete(0, tk.END)
MAINEXCLUDE.delete(0, tk.END)
# First see if it exists
if (not path.exists("exclude.json")):
    # Create it
    with open("exclude.json", 'w') as file:
        excludeDict = {}

else:
    with open("exclude.json", 'r') as file:
        excl = json.load(file)
        for name, value in excl.items():
            entry = '{"' + name + '":"' + value + '"'
            newentry = eval(entry)
            excludeDict.update(newentry)
            EXCLUDE.insert(tk.END, f'"{name}", "{value}"')
            MAINEXCLUDE.insert(tk.END, f'"{name}", "{value}"')

        subCount.set(len(excludeDict))
        subMessage.set("JSON loaded")
# End readExclude

### GUI
sub = tk.Toplevel()
sub.title("Sub-Directory Sync")
sub.config(background = subDirColor.get())

# Frames
subentryframe = tk.LabelFrame(sub, text = "Sub Directory Entries", background = subDirColor.get())
subentryframe.grid(row = 0, column = 0, columnspan = 5, sticky = ('e', 'w'))

submsgframe = tk.LabelFrame(sub, text = "Messages", background = subDirColor.get())
submsgframe.grid(row = 1, column = 0, columnspan = 5, sticky = ('e', 'w'))

subcmdframe = tk.LabelFrame(sub, text = "Commands", background = subDirColor.get())
subcmdframe.grid(row = 2, column = 0, columnspan = 5, sticky = ('e', 'w'))

# tkinter Variable
subMessage = tk.StringVar()
toExclude = tk.StringVar()
subCount = tk.IntVar()

# Initializations

```

```

# Entry Area

tk.Label(subentryframe, text = "Enter a 'Name' for the item and the 'name/extension'", background =
subDirColor.get(), foreground = "green").grid(row = 0, column = 0, columnspan = 3, sticky = ('w'))

tk.Label(subentryframe, text = "File or Pattern to Exclude:", background = subDirColor.get()).grid(row
= 1, column = 1, sticky = ('e'))

TOEXCLUDE = tk.Entry(subentryframe, textvariable = toExclude, background = "lightblue", width = 30)
TOEXCLUDE.grid(row = 1, column = 2, sticky = ('w'))

SUBBROWSE = tk.Button(subentryframe, text = "Browse", foreground = "green", activeforeground = "green",
width = 6,

                        cursor = "hand2", command = subBrowse)

SUBBROWSE.grid(row = 1, column = 4, sticky = ('w'))

tk.Label(subentryframe, text = ' Ex: "Text":"*.txt" OR use "Browse"', background = subDirColor.get(),
foreground = "red").grid(row = 2, column = 2, columnspan = 3, sticky = ('w'))

tk.Label(subentryframe, text = "Destination Directory:", background = subDirColor.get()).grid(row = 3,
column = 1, sticky = ('e'))

SUBDESTINATION = tk.Entry(subentryframe, textvariable = subDestination, background = "lightblue", width
= 30)

SUBDESTINATION.grid(row = 3, column = 2, sticky = ('w'))

SUBDESTBROWSE = tk.Button(subentryframe, text = "Browse", foreground = "green", activeforeground =
"green", width = 6,

                        cursor = "hand2", command = subDestBrowse)

SUBDESTBROWSE.grid(row = 3, column = 4, sticky = ('w'))

tk.Label(subentryframe, text = "Add Exclude Directory:", background = subDirColor.get()).grid(row = 4,
column = 1, sticky = ('e'))

EXCLUDEDIRECTORY = tk.Entry(subentryframe, textvariable = excludeDirectory, background = "lightblue",
width = 30)

EXCLUDEDIRECTORY.grid(row = 4, column = 2, sticky = ('w'))

SUBDIRBROWSE = tk.Button(subentryframe, text = "Browse", foreground = "green", activeforeground =
"green", width = 6,

                        cursor = "hand2", command = subDirBrowse)

SUBDIRBROWSE.grid(row = 4, column = 4, sticky = ('w'))

tk.Label(subentryframe, text = "Count:", background = subDirColor.get()).grid(row = 5, column = 0,
sticky = ('e'))

SUBCOUNT = tk.Label(subentryframe, textvariable = subCount, background = "lightyellow", width = 3)
SUBCOUNT.grid(row = 5, column = 1, sticky = ('w'))

EXCLUDE = tk.Listbox(subentryframe, background = "lightyellow", width = 50, height = 10)
EXCLUDE.grid(row = 6, column = 0, columnspan = 4, sticky = ('w'))

# Message Area

tk.Label(submsgframe, text = "Message:", background = subDirColor.get()).grid(row = 0, column = 0,
sticky = ('e'))

SUBMESSAGE = tk.Label(submsgframe, textvariable = subMessage, background = "lightyellow", width = 60)
SUBMESSAGE.grid(row = 0, column = 1, sticky = ('w'))

# Command Area

SUBCLOSE = tk.Button(subcmdframe, text = "Close", foreground = "red", activeforeground = "red", width =
5,

```

```

        cursor = "hand2", command = subClose)

SUBCLOSE.grid(row = 0, column = 0, sticky = ('w'))

ADDITEM = tk.Button(subcmdframe, text = "Add Item", foreground = "green", activeforeground = "green",
width = 10,

        cursor = "hand2", command = addItem)

ADDITEM.grid(row = 0, column = 1, sticky = ('w'))

REMOVEITEM = tk.Button(subcmdframe, text = "Remove Item", foreground = "red", activeforeground = "red",
width = 10,

        cursor = "hand2", command = removeItem)

REMOVEITEM.grid(row = 0, column = 2, sticky = ('w'))

# Immediate Function Calls
clearSubMessage()
readExclude()
TOEXCLUDE.focus()

# Bindings
TOEXCLUDE.bind("<Return>", doAddItem)

# Global Configurations
for child in subentryframe.winfo_children():child.grid_configure(padx=2, pady=2)
for child in submsgframe.winfo_children():child.grid_configure(padx=2, pady=2)
for child in subcmdframe.winfo_children():child.grid_configure(padx=2, pady=2)

sub.mainloop()
# End openSubDirectoryCopy

def openConfig() -> None:
    """This will open the configuration window."""
    def configClose() -> None:
        """This function closes the configuration window."""
        conf.destroy()
    # End configClose

def clearConfMessage() -> None:
    """This clears the message variable and starts the timer."""
    confmessage.set("")
    conf.after(15000, clearConfMessage)
# End clearConfMessage

def snglBrowse() -> None:
    """This opens the filedialog for user input
    to choose the source directory path."""
    # I need to add in the size of the selected drive
    sng = filedialog.askdirectory()
    if (sng != ""):

```

```

        snglDirectory.set(sng)
        getDiskFreeSpace(sng)
# End snglBrowse

def snglDestBrowse() -> None:
    """This opens the filedialog for user input."""
    # I need to add in the size of the selected drive
    sngDest = filedialog.askdirectory()
    if (sngDest != ""):
        snglDestination.set(sngDest)
        getDiskFreeSpace(sngDest)
# End snglDestBrowse

def subBrowse() -> None:
    """This opens the filedialog for user input."""
    # I need to add in the size of the selected drive
    sub = filedialog.askdirectory()
    if (sub != ""):
        subDirectory.set(sub)
        getDiskFreeSpace(sub)
# End subBrowse

def subDestBrowse() -> None:
    """This opens the filedialog for user input."""
    # I need to add in the size of the selected drive
    subDest = filedialog.askdirectory()
    if (subDest != ""):
        subDestination.set(subDest)
        getDiskFreeSpace(subDest)
# End subDestBrowse

def saveConfig() -> None:
    """This function writes out the configuration data."""
    # Create the JSON
    confJson = {"snglDirectory":f"{snglDirectory.get()}",
                "snglDestination":f"{snglDestination.get()}",
                "subDirectory":f"{subDirectory.get()}",
                "subDestination":f"{subDestination.get()}",
                "useRecursion":f"{useRecursion.get()}",
                "PDFViewer":f"{PDFViewer.get()}",
                "startTime":f"{startTime.get()}",
                "stopTime":f"{stopTime.get()}",
                "useSchedule":f"{useSchedule.get()}",
                "logPath":f"{logPath.get()}",
                "singleDirColor":f"{singleDirColor.get()}",
                "subDirColor":f"{subDirColor.get()}"}

```

```

with open("config.json", 'w') as file:
    json.dump(confJson, file)

loadConfig()

confmessage.set("Configuration saved")
# End saveConfig

def browsePDFViewer() -> None:
    """This function opens the filedialog to
    get the program that will be used to
    view the Help PDF file."""
    pdfviewer = filedialog.askopenfilename()
    if (pdfviewer != ""):
        PDFViewer.set(pdfviewer)
# End browsePDFViewer

def snglColorChooser() -> None:
    """This function selects the color of the
    single director window."""
    snglColor = askcolor(title = "Pick a Color")
    if (snglColor == (None, None)):
        return
    singleDirColor.set(snglColor[1])
    SNGLCOLOR.config(background = snglColor[1])
# End snglColorChooser

def subColorChooser() -> None:
    """This function selects the color of the
    sub directory window."""
    subColor = askcolor(title = "Pick a Color")
    if (subColor == (None, None)):
        return
    subDirColor.set(subColor[1])
    SUBCOLOR.config(background = subColor[1])
# End subColorChooser

def browseLogPath() -> None:
    """This function uses the filedialog to
    select the logfile to use."""
    logP = filedialog.askopenfilename()
    if (logP != ""):
        logPath.set(logP)
# End browseLogPath

## GUI

```



```

conf = tk.Toplevel()
conf.title("System configuration")
conf.config(background = "tan")

# Frames
dataframe = tk.LabelFrame(conf, text = "Configuration", background = "tan")
dataframe.grid(row = 0, column = 0, columnspan = 6, sticky = ('e'))

colorframe = tk.LabelFrame(conf, text = "Window Colors", background = "tan")
colorframe.grid(row = 1, column = 0, columnspan = 6, sticky = ('e','w'))

confmsgframe = tk.LabelFrame(conf, text = "Messages", background = "tan")
confmsgframe.grid(row = 2, column = 0, columnspan = 6, sticky = ('e','w'))

confcmdframe = tk.LabelFrame(conf, text = "Commands", background = "tan")
confcmdframe.grid(row = 3, column = 0, columnspan = 6, sticky = ('e','w'))

# Variables
confmessage          = tk.StringVar()

# Data Area
tk.Label(dataframe, text = "Single-Directory Path:", background = "tan").grid(row = 0, column = 0,
sticky = ('e'))
SNGLDIRECTORY = tk.Entry(dataframe, textvariable = snglDirectory, background = "lightblue", width = 40)
SNGLDIRECTORY.grid(row = 0, column = 1, columnspan = 2, sticky = ('w'))
SNGLBROWSE = tk.Button(dataframe, text = "Browse", foreground = "green", activeforeground = "green",
width = 6,
                        cursor = "hand2", command = snglBrowse)
SNGLBROWSE.grid(row = 0, column = 4, sticky = ('w'))

tk.Label(dataframe, text = "Single-Destination:", background = "tan").grid(row = 1, column = 0, sticky
= ('e'))
SNGLDESTINATION = tk.Entry(dataframe, textvariable = snglDestination, background = "lightblue", width =
40)
SNGLDESTINATION.grid(row = 1, column = 1, columnspan = 2, sticky = ('w'))
SNGLDESTBROWSE = tk.Button(dataframe, text = "Browse", foreground = "green", activeforeground =
"green", width = 6,
                        cursor = "hand2", command = snglDestBrowse)
SNGLDESTBROWSE.grid(row = 1, column = 4, sticky = ('w'))

tk.Label(dataframe, text = "Sub-Directory Path:", background = "tan").grid(row = 2, column = 0, sticky
= ('e'))
SUBDIRECTYORY = tk.Entry(dataframe, textvariable = subDirectory, background = "lightblue", width = 40)
SUBDIRECTYORY.grid(row = 2, column = 1, columnspan = 2, sticky = ('w'))
SUBBROWSE = tk.Button(dataframe, text = "Browse", foreground = "green", activeforeground = "green",
width = 6,
                        cursor = "hand2", command = subBrowse)
SUBBROWSE.grid(row = 2, column = 4, sticky = ('w'))

tk.Label(dataframe, text = "Sub-Destination:", background = "tan").grid(row = 3, column = 0, sticky =

```

```

('e'))
SUBDESTINATION = tk.Entry(dataframe, textvariable = subDestination, background = "lightblue", width =
40)
SUBDESTINATION.grid(row = 3, column = 1, columnspan = 2, sticky = ('w'))
SUBDESTBROWSE = tk.Button(dataframe, text = "Browse", foreground = "green", activeforeground = "green",
width = 6,
                           cursor = "hand2", command = subDestBrowse)
SUBDESTBROWSE.grid(row = 3, column = 4, sticky = ('w'))

tk.Label(dataframe, text = "PDF Viewer:", background = "tan").grid(row = 4, column = 0, sticky = ('e'))
PDFVIEWER = tk.Entry(dataframe, textvariable = PDFViewer, background = "lightblue", width = 30)
PDFVIEWER.grid(row = 4, column = 1, columnspan = 2, sticky = ('w'))
BROWSEPDFVIEWER = tk.Button(dataframe, text = "Browse", foreground = "green", activeforeground =
"green", width = 6,
                             cursor = "hand2", command = browsePDFViewer)
BROWSEPDFVIEWER.grid(row = 4, column = 4, sticky = ('w'))

tk.Label(dataframe, text = "Copy Start Time:", background = "tan").grid(row = 5, column = 0, sticky =
('e'))
COPYSTARTTIME = tk.Entry(dataframe, textvariable = startTime, background = "lightblue", width = 5)
COPYSTARTTIME.grid(row = 5, column = 1, sticky = ('w'))
tk.Label(dataframe, text = "24-Hour format without colons", background = "tan").grid(row = 5, column =
2, sticky = ('w'))

tk.Label(dataframe, text = "Copy Stop Time:", background = "tan").grid(row = 6, column = 0, sticky =
('e'))
COPYSTOPTIME = tk.Entry(dataframe, textvariable = stopTime, background = "lightblue", width = 5)
COPYSTOPTIME.grid(row = 6, column = 1, sticky = ('w'))
tk.Label(dataframe, text = "24-Hour format without colons", background = "tan").grid(row = 6, column =
2, sticky = ('w'))

ENABLESCHEDULE = tk.Checkbutton(dataframe, text = "Enable Scheduleing", variable = useSchedule,
background = "tan",
                                activebackground = "tan", offvalue = 0, onvalue = 1, cursor = "hand2")
ENABLESCHEDULE.grid(row = 7, column = 1, columnspan = 3, sticky = ('w'))

tk.Label(dataframe, text = "Log Path:", background = "tan").grid(row = 8, column = 0, sticky = ('e'))
LOGPATH = tk.Entry(dataframe, textvariable = logPath, background = "lightblue", width = 40)
LOGPATH.grid(row = 8, column = 1, columnspan = 2, sticky = ('w'))
LOGPATHBROWSE = tk.Button(dataframe, text = "Browse", foreground = "green", activeforeground = "green",
width = 6,
                           cursor = "hand2", command = browseLogPath)
LOGPATHBROWSE.grid(row = 8, column = 4, sticky = ('w'))

# Color Area
tk.Label(colorframe, text = "Single-Directory Color:", background = "tan").grid(row = 0, column = 0,
sticky = ('e'))
SNGLCOLOR = tk.Entry(colorframe, textvariable = singleDirColor, background = singleDirColor.get(),
width = 8)
SNGLCOLOR.grid(row = 0, column = 1, sticky = ('w'))
SNGLCOLORBROWSE = tk.Button(colorframe, text = "Color", foreground = "blue", activeforeground = "blue",
width = 6,

```

```

        cursor = "hand2", command = snglColorChooser)

    SNGLCOLORBROWSE.grid(row = 0, column = 2, sticky = ('w'))

    tk.Label(colorframe, text = "Sub-Directory Color:", background = "tan").grid(row = 0, column = 3,
    sticky = ('e'))

    SUBCOLOR = tk.Entry(colorframe, textvariable = subDirColor, background = subDirColor.get(), width = 8)

    SUBCOLOR.grid(row = 0, column = 4, sticky = ('w'))

    SUBCOLORBROWSE = tk.Button(colorframe, text = "Color", foreground = "blue", activeforeground = "blue",
    width = 6,

        cursor = "hand2", command = subColorChooser)

    SUBCOLORBROWSE.grid(row = 0, column = 5, sticky = ('w'))

    # Message Area

    tk.Label(confmsgframe, text = "Message:", background = "black", foreground = "yellow").grid(row = 0,
    column = 0, sticky = ('e'))

    CONFMESSAGE = tk.Label(confmsgframe, textvariable = confmessage, background = "lightyellow", width =
    30)

    CONFMESSAGE.grid(row = 0, column = 1, columnspan = 3, sticky = ('w'))

    # Command Area

    CONFIGCLOSE = tk.Button(confcmdframe, text = "Close", foreground = "red", activeforeground = "red",
    width = 7,

        cursor = "hand2", command = configClose)

    CONFIGCLOSE.grid(row = 0, column = 0, sticky = ('w'))

    SAVECONFIG = tk.Button(confcmdframe, text = "Save Config", foreground = "green", activeforeground =
    "green", width = 10,

        cursor = "hand2", command = saveConfig)

    SAVECONFIG.grid(row = 0, column = 2, sticky = ('w'))

    RELOADCONFIG = tk.Button(confcmdframe, text = "Reload Config", foreground = "blue", activeforeground =
    "blue", width = 10,

        cursor = "hand2", command = loadConfig)

    RELOADCONFIG.grid(row = 0, column = 3, sticky = ('w'))

    # Immediate Functions

    clearConfMessage()

    loadConfig()

    # Bindings

    SNGLDIRECTORY.bind("<FocusOut>", doLoadConfig)

    SNGLDESTINATION.bind("<FocusOut>", doLoadConfig)

    SUBDIRECTORY.bind("<FocusOut>", doLoadConfig)

    SUBDESTINATION.bind("<FocusOut>", doLoadConfig)

    for child in dataframe.wininfo_children():child.grid_configure(padx = 2, pady = 2)

    for child in colorframe.wininfo_children():child.grid_configure(padx = 2, pady = 2)

    for child in confmsgframe.wininfo_children():child.grid_configure(padx = 2, pady = 2)

```

```

        for child in confcmdframe.wininfo_children():child.grid_configure(padx = 7, pady = 2)

    conf.mainloop()
# End openConfig

def openHelp() -> None:
    """Opens the help PDF document."""
    try:
        subprocess.Popen([PDFViewer.get(), 'FileCopyHelp.pdf'])

    except AttributeError:
        theMessage.set("openHelp Error")

# End openHelp

def openAbout() -> None:
    """Opens the "About" form showing the current version
    of the software as well as author information."""
    def doClose():
        about.destroy()

    about = tk.Toplevel()
    about.title("About")
    theWords = f"directorySync.py\nWritten by Thomas C. Smith\n{revDate.get()}\nVersion: {version.get()}"
    theMessage = tk.Message(about, text=theWords,width=250, justify=tk.CENTER)
    theMessage.grid(column=0, row=0, columnspan=4, padx=5, pady=10)
    button = tk.Button(about, text = "Close", foreground = "red", activeforeground = "red",
                       cursor = "hand2", command=doClose).grid(column=0, row=1)

# End openAbout

def checkSchedule() -> None:
    """This function checks to see if we are to use
    the scheduled times and if we are in that time window."""
    if (useSchedule.get() == 0):
        # Just bail out
        return

    else:
        # Check the time window to see if we are within that window
        T = time.localtime()
        H = str(T.tm_hour)
        if (len(H) < 2):
            H = "0" + H
        M = str(T.tm_min)
        if (len(M) < 2):

```

```

        M = "0" + M

    Tnow = H + M

    if (startTime.get() >= TNow and stopTime.get() <= Tnow):
        # Yes, within the window so start copytree
        subMenu.set("Starting Single Directory Sync")

    else:
        subMessage.set("Not in the time window")

# End checkSchedule

def startSingleSync() -> None:
    """This function starts or stops the
    single-directory synchronizing and
    modified the text of the button to
    show either Start or Stop."""
    # With the single directory we use the copy2 method of the shutil module
    message.set("Starting Single Sync")
    copyList = []
    message.set("Synchronizing")
    curList = listdir(snglDirectory.get())
    oldList = listdir(snglDestination.get())
    for file in curList:
        if (path.isdir(file)):
            continue

        # Here we hardcode in the default FITS forms and add the includeDict dictionary
        if ((file not in oldList) and ((file.endswith(".fits")
                                         or (file.endswith(".fit")
                                              or (file.endswith(".fts")
                                                  or (file in includeDict)))))):
            copyList .append(file)

    if (copyList == []):
        message.set("Nothing new to copy")

    else:
        for cFile in copyList:
            src = snglDirectory.get() + "/" + cFile
            dst = snglDestination.get() + "/" + cFile
            message.set(f"Copying: {cFile}")
            shutil.copy2(src,dst) # Using copy2 maintains the file's metadata
            logIt(f"{makeTimestamp('full')} : COPY : {cFile} Copied from {src} to {dst}")
            message.set("Copy complete")

# End startSingleSync

def startSubSync() -> None:

```

```

"""This function starts or stops the sub-directory
syncing and modified the button to show either
start or stop."""
global excludeDict, includeDict
# With the sub directory we use the copytree method of the shutil module
# 'copytree' method uses the copy2 form so metadata of the file is preserved
message.set("Starting Sub Sync")
# Work with the excludeDict.values
# Change into the destination directory and work from inside there
chdir(subDestination.get())
# Now do the copytree using just the timestamp name
vals = [] # We use the 'unpacker *' in the ignore_patterns method
for item in excludeDict.values():
    vals.append(item)

inc = 0
src = subDirectory.get()
dest = subDestination.get() + "/" + makeTimestamp()
# Check to see if the dest directory exists and if so prompt asking to overwrite it
if (path.exists(dest)):
    # Yes it already exists so ask
    mAns = messagebox.askyesno("Overwrite the Directory",
                                "The Directory already exists,/ndo you want to overwrite it?")

    if (mAns == True):
        chdir("/home/pi/VirtualEnv/newCopyFile/fileCopy/test2/")
        try:
            shutil.rmtree(makeTimestamp())

        except:
            message.set("Error removing directory")
            return

    else:
        return

try:
    shutil.copytree(src, dest, ignore = shutil.ignore_patterns(*vals))
    # Return to the source directory
    chdir(src)
    # Now log it
    logIt(f"{makeTimestamp('full')} : COPY : {src} COPIED TO {dest}")
    message.set("Copy complete")

except:
    message.set("File error has occurred")

```

```

# End startSubSync

def doLoadConfig(event) -> None:
    loadConfig()
# End doLoadConfig

def loadConfig() -> None:
    """Loads the configuration file."""
    with open("config.json", 'r') as file:
        confJson = json.load(file)

    # Now populate the variables
    snглDirectory.set(confJson["snглDirectory"])
    snглDestination.set(confJson["snглDestination"])
    subDirectory.set(confJson["subDirectory"])
    subDestination.set(confJson["subDestination"])
    if (subDestination.get().find("_202")):
        # See if the timestamp is the same and if so then add incrementer digit
        pass

    else:
        subDestination.set(subDestination.get() + "_" + makeTimestamp())
    useRecursion.set(confJson["useRecursion"])
    PDFViewer.set(confJson["PDFViewer"])
    startTime.set(confJson["startTime"])
    stopTime.set(confJson["stopTime"])
    useSchedule.set(confJson["useSchedule"])
    logPath.set(confJson["logPath"])
    singleDirColor.set(confJson["singleDirColor"])
    subDirColor.set(confJson["subDirColor"])
    # Get disk sizes
    errMessage = ""
    try:    # Overall
        try:    # Single Directory (SOURCE)
            snглDirSize.set(getDiskFreeSpace(snглDirectory.get()))

        except FileNotFoundError as e:
            errMessage = "Single Source"
            snглDirSize.set("UNK")

        try:    # Single Destination
            snглDestSize.set(getDiskFreeSpace(snглDestination.get()))

        except FileNotFoundError as e:

```

```

        errMessage = errMessage + "Single Destination"
        snglDestSize.set("UNK")

    try:        # Sub Directory (SOURCE)
        subDirSize.set(getDiskFreeSpace(subDirectory.get()))

    except FileNotFoundError as e:
        errMessage = errMessage + "Sub Source"
        subDirSize.set("UNK")

    try:        # Sub Destination
        D = subDestination.get()
        subD = subDestination.get().find("_202")
        if (subD >= 0):
            nSubD = D[:subD]

        else:
            nSubD = D

        subDestSize.set(getDiskFreeSpace(nSubD))

    except FileNotFoundError as e:
        errMessage = errMessage + "Sub Destination"

except FileNotFoundError as e:
    message.set(errMessage + " File or Drive not found")

if (errMessage == ""):
    STARTSINGLE.config(state = "normal")
    STARTSUB.config(state = "normal")

else:
    if (errMessage.find("Single") >= 0):
        STARTSINGLE.config(state = "disabled")

    else:
        STARTSINGLE.config(state = "normal")

    if (errMessage.find("Sub") >= 0):
        STARTSUB.config(state = "disabled")

    else:
        STARTSUB.config(state = "normal")

# End loadConfig

```



```

def formatSize(bytes) -> str:
    """Calculates the size and units for
    the passed in byte count."""
    try:
        bytes = float(bytes)
        kb = bytes / 1024

    except:
        return "Error"

    if kb >= 1024:
        M = kb / 1024
        if M >= 1024:
            G = M / 1024
            return "%.2fG" % (G)

        else:
            return "%.2fM" % (M)

    else:
        return "%.2fkb" % (kb)
# End formatSize

def getDiskFreeSpace(disk : str) -> str:
    """ Return disk free space (in bytes)
    """
    if platform.system() == 'Windows':
        free_bytes = ctypes.c_ulonglong(0)
        ctypes.windll.kernel32.GetDiskFreeSpaceExW(ctypes.c_wchar_p(disk),
                                                    None, None, ctypes.pointer(free_bytes))
        return formatSize(free_bytes.value)

    else:
        st = os.statvfs(disk)
        return formatSize(st.f_bavail * st.f_frsize)

# Example:
# print(getDiskFreeSpace("F:\\"))
# End getDiskFreeSpace

def openLogManager() -> None:
    """This window manages the file copies
    logging."""
    def closeLog() -> None:
        """This closes the log manager window."""
        log.destroy()

```

```

# End closeLog

def clearLogMessage() -> None:
    """This function clears the log and
    starts the auto-run timer."""
    logMessage.set("")
    log.after(15000, clearLogMessage)
# End clearLogMessage

def getLogFiles() -> None:
    """This function reads the log files
    present in the program directory and
    populates them into the list 'logFiles.'"""
    global logFiles
    # This looks for files with the extension og '.log'
    arrL = logPath.get().split("/")
    logToUse.set(arrL[-1])
    lenFilename = len(logToUse.get())
    totalL = len(logPath.get()) - len(logToUse.get())
    shortPath = logPath.get()[:totalL]
    lp = listdir(shortPath)
    for items in lp:
        if (items[-3:] == "log"):
            logFiles.append(items)

    with open(logPath.get(), 'r') as file:
        entry = file.readlines()
        if (len(entry) == 0):
            with open(logPath.get(), 'w') as file:
                LOGDATA.delete(0, tk.END)
                logText =
"-----\n"
                logText = logText + makeTimestamp('full') + " : New Log Created\n"
                logText = logText +
"-----\n"
                file.write(logText)
                for i in range(3):
                    LOGDATA.insert(tk.END, logText[i])

            else:
                for i in range(len(entry)):
                    LOGDATA.insert(tk.END, entry[i].strip())
    refreshLog()
# End getLogFiles

def selectLogFile(event) -> None:
    """This function selects the logfile to use."""

```

```

        logToUse.set(LOGFILE.get())
# End selectLogFile

def archiveLog() -> None:
    """This function archives the current log
    file and creates a new file."""
    pass
# End archiveLog

def refreshLog() -> None:
    """This reloads the log file."""
    LOGDATA.delete(0, tk.END)
    with open(logPath.get(), 'r') as file:
        entries = file.readlines()
        for line in entries:
            LOGDATA.insert(tk.END, line.strip())

    log.after(5000, refreshLog)
# End refreshLog

### GUI
log = tk.Toplevel()
log.title("Log Manager")
log.config(background = "tan")

# Frames
loginfoframe = tk.LabelFrame(log, text = "Log Information", background = "tan")
loginfoframe.grid(row = 0, column = 0, columnspan = 4, sticky = ('e', 'w'))

logdataframe = tk.LabelFrame(log, text = "Log Data", background = "tan")
logdataframe.grid(row = 1, column = 0, columnspan = 4, sticky = ('e', 'w'))

logmsgframe = tk.LabelFrame(log, text = "Messages", background = "tan")
logmsgframe.grid(row = 2, column = 0, columnspan = 4, sticky = ('e', 'w'))

logcmdframe = tk.LabelFrame(log, text = "Commands", background = "tan")
logcmdframe.grid(row = 3, column = 0, columnspan = 4, sticky = ('e', 'w'))

# tkinter Variable
logMessage      = tk.StringVar()

# Inialization

# Log Info Area

```

```

tk.Label(loginframe, text = "Log File:", background = "tan").grid(row = 0, column = 0, sticky =
('e'))

LOGFILE = ttk.Combobox(loginframe, values = logFiles, textvariable = logToUse, state = "readonly")
LOGFILE.grid(row = 0, column = 1, sticky = ('w'))

# Log Data Area
LOGDATA = tk.Listbox(logdataframe, background = "lightyellow", width = 80, height = 10)
LOGDATA.grid(row = 0, column = 0, columnspan = 3, sticky = ('e','w'))

# Message Area
tk.Label(logmsgframe, text = "Message:", background = "black", foreground = "yellow").grid(row = 0,
column = 0, sticky = ('e'))
LOGMESSAGE = tk.Label(logmsgframe, textvariable = logMessage, background = "lightyellow", width = 40)
LOGMESSAGE.grid(row = 0, column = 1, columnspan = 3, sticky = ('w'))

# Command Area
CLOSELOG = tk.Button(logcmdframe, text = "Close", foreground = "red", activeforeground = "red", width =
6,
cursor = "hand2", command = closeLog)
CLOSELOG.grid(row = 0, column = 0, sticky = ('w'))

REFRESHLOG = tk.Button(logcmdframe, text = "Refresh", foreground = "blue", activeforeground = "blue",
width = 8,
cursor = "hand2", command = refreshLog)
REFRESHLOG.grid(row = 0, column = 1, sticky = ('w'))

# Immediate Function Calls
getLogFiles()
clearLogMessage()

# Bindings
LOGFILE.bind("<<ComboboxSelected>>", selectLogFile)

# Global Layouts
for child in loginframe.winfo_children():child.grid_configure(padx = 2, pady = 2)
for child in logdataframe.winfo_children():child.grid_configure(padx = 2, pady = 2)
for child in logmsgframe.winfo_children():child.grid_configure(padx = 2, pady = 2)
for child in logcmdframe.winfo_children():child.grid_configure(padx = 2, pady = 2)

log.mainloop()

# End openLogManager

### End of Support Functions / Methods

### GUI
sync = tk.Tk()
sync.title("File / Directory Synchronization")
sync.config(background = "tan")

```

```

# Frames

mainframe = tk.Frame(sync, background = "tan")
mainframe.grid(row = 0, column = 0, columnspan = 4, sticky = ('e','w'))

schedframe = tk.LabelFrame(mainframe, text = "Schedule", background = "tan")
schedframe.grid(row = 1, column = 0, columnspan = 5, sticky = ('e','w'))

dispframe = tk.LabelFrame(mainframe, text = "Current Settings", background = "tan")
dispframe.grid(row = 2, column = 0, columnspan = 3, sticky = ('e','w'))

snglframe = tk.LabelFrame(dispframe, text = "Single Directory  Parm", background = "tan")
snglframe.grid(row = 0, column = 0, columnspan = 2, sticky = ('e','w'))

subframe = tk.LabelFrame(dispframe, text = "Sub Directory Parm", background = "tan")
subframe.grid(row = 0, column = 2, columnspan = 2, sticky = ('e','w'))

diskframe = tk.LabelFrame(mainframe, text = "Disk Sizes & Paths", background = "tan")
diskframe.grid(row = 3, column = 0, columnspan = 4, sticky = ('e','w'))

msgframe = tk.LabelFrame(mainframe, text = "Messages", background = "tan")
msgframe.grid(row = 4, column = 0, columnspan = 4, sticky = ('e','w'))

cmdframe = tk.LabelFrame(mainframe, text = "Commands", background = "tan")
cmdframe.grid(row = 5, column = 0, columnspan = 4, sticky = ('e','w'))

# Menu

mainmenu = tk.Menu(sync)

# File Menu

filemenu = tk.Menu(mainmenu, tearoff = 0)
filemenu.add_command(label = "Quit", command = doQuit)
mainmenu.add_cascade(label = "File", menu = filemenu)

# single Menu

singlemenu = tk.Menu(mainmenu, tearoff = 0)
singlemenu.add_command(label = "Single-Directory Sync", command = openSingleDirectoryCopy)
mainmenu.add_cascade(label = "Single", menu = singlemenu)

# subdirectory Menu

subdirectorymenu = tk.Menu(mainmenu, tearoff = 0)
subdirectorymenu.add_command(label = "Subdirectory-Sync", command = openSubDirectoryCopy)
mainmenu.add_cascade(label = "Subdirectory", menu = subdirectorymenu)

# Configuration Menu

configmenu = tk.Menu(mainmenu, tearoff = 0)
configmenu.add_command(label = "Config Manager", command = openConfig)

```

```

mainmenu.add_cascade(label = "Configuration", menu = configmenu)

# Log Menu
logmenu = tk.Menu(mainmenu, tearoff = 0)
logmenu.add_command(label = "Log Manager", command = openLogManager)
mainmenu.add_cascade(label = "Logs", menu = logmenu)

helpmenu = tk.Menu(mainmenu, tearoff = 0)
helpmenu.add_command(label = "Help PDF", command = openHelp)
helpmenu.add_command(label = "About", command = openAbout)
mainmenu.add_cascade(label = "Help", menu = helpmenu)

sync.config(menu = mainmenu)

# tkinter variables
message = tk.StringVar()
startTime = tk.StringVar()
stopTime = tk.StringVar()
useSchedule = tk.IntVar() # Flag indicating if the program should run during the scheduled
times
snglDirectory = tk.StringVar()
snglDirSize = tk.StringVar()
snglDestination = tk.StringVar()
snglDestSize = tk.StringVar()
subDirectory = tk.StringVar()
subDirSize = tk.StringVar()
subDestination = tk.StringVar()
subDestSize = tk.StringVar()
excludeDirectory = tk.StringVar()
useRecursion = tk.IntVar()
PDFViewer = tk.StringVar()
version = tk.StringVar()
revDate = tk.StringVar()
singleDirColor = tk.StringVar() # User selectable color
subDirColor = tk.StringVar() # User selectable color
logPath = tk.StringVar()
logToUse = tk.StringVar()

# Initialization
version.set("1.0.0")
revDate.set("April 22, 2023")
singleDirColor.set("lightgreen")
subDirColor.set("lightgray")

# Schedule Area
tk.Label(schedframe, text = "Start:", background = "tan").grid(row = 0, column = 0, sticky = ('e'))
STARTTIME = tk.Entry(schedframe, textvariable = startTime, background = "lightblue", width = 5)

```

```

STARTTIME.grid(row = 0, column = 1, sticky = ('w'))

tk.Label(schedframe, text = "Stop:", background = "tan").grid(row = 0, column = 2, sticky = ('e'))
STOPTIME = tk.Entry(schedframe, textvariable = stopTime, background = "lightblue", width = 5)
STOPTIME.grid(row = 0, column = 3, sticky = ('w'))

USESCHEDULE = tk.Checkbutton(schedframe, text = "Use Schedule", variable = useSchedule, onvalue = 1,
offvalue = 0,
                                background = "tan", activebackground = "tan", cursor = "hand2", command =
checkSchedule)
USESCHEDULE.grid(row = 0, column = 4, sticky = ('w'))

## Display Area
# Single Area
MAININCLUDE = tk.Listbox(snglframe, background = "lightyellow", width = 30, height = 10)
MAININCLUDE.grid(row = 0, column = 0, columnspan = 2, rowspan = 10, sticky = ('e','w'))

# Sub Area
MAINEXCLUDE = tk.Listbox(subframe, background = "lightyellow", width = 30, height = 10)
MAINEXCLUDE.grid(row = 0, column = 0, columnspan = 2, rowspan = 10, sticky = ('e','w'))

# Disk Size Area
tk.Label(diskframe, text = "Single-Source Drive:", background = "tan").grid(row = 0, column = 0, sticky =
('e'))
SNGLSOURCESIZE = tk.Label(diskframe, textvariable = snglDirSize, background = "lightyellow", width = 10)
SNGLSOURCESIZE.grid(row = 0, column = 1, sticky = ('w'))

tk.Label(diskframe, text = "Single-Dest Drive:", background = "tan").grid(row = 0, column = 2, sticky =
('e'))
SNGLDESTSIZE = tk.Label(diskframe, textvariable = snglDestSize, background = "lightyellow", width = 10)
SNGLDESTSIZE.grid(row = 0, column = 3, sticky = ('w'))

tk.Label(diskframe, text = "Sub-Source Drive:", background = "tan").grid(row = 1, column = 0, sticky =
('e'))
SUBSOURCESIZE = tk.Label(diskframe, textvariable = subDirSize, background = "lightyellow", width = 10)
SUBSOURCESIZE.grid(row = 1, column = 1, sticky = ('w'))

tk.Label(diskframe, text = "Sub-Dest Drive:", background = "tan").grid(row = 1, column = 2, sticky = ('e'))
SUBDESTSIZE = tk.Label(diskframe, textvariable = subDestSize, background = "lightyellow", width = 10)
SUBDESTSIZE.grid(row = 1, column = 3, sticky = ('w'))

tk.Label(diskframe, text = "Single-Directory Source:", background = "tan").grid(row = 2, column = 0, sticky
= ('e'))
SNGLSOURCE = tk.Label(diskframe, textvariable = snglDirectory, background = "lightyellow", width = 40,
anchor = "w")
SNGLSOURCE.grid(row = 2, column = 1, columnspan = 3, sticky = ('w'))

tk.Label(diskframe, text = "Single-Directory Destination:", background = "tan").grid(row = 3, column = 0,
sticky = ('e'))
SNGLDESTINATION = tk.Label(diskframe, textvariable = snglDestination, background = "lightyellow", width =
40, anchor = "w")

```

```

SNGLDESTINATION.grid(row = 3, column = 1, columnspan = 3, sticky = ('w'))

tk.Label(diskframe, text = "Sub-Directory Source:", background = "tan").grid(row = 4, column = 0, sticky =
('e'))

SUBSOURCE = tk.Label(diskframe, textvariable = subDirectory, background = "lightyellow", width = 40, anchor
= "w")

SUBSOURCE.grid(row = 4, column = 1, columnspan = 3, sticky = ('w'))

tk.Label(diskframe, text = "Sub-Directory Destination:", background = "tan").grid(row = 5, column = 0,
sticky = ('e'))

SUBDESTINATION = tk.Label(diskframe, textvariable = subDestination, background = "lightyellow", width = 40,
anchor = "w")

SUBDESTINATION.grid(row = 5, column = 1, columnspan = 3, sticky = ('w'))

# Message area

tk.Label(msgframe, text = "Message:", background = "black", foreground = "yellow").grid(row = 0, column =
0, sticky = ('e'))

MESSAGE = tk.Label(msgframe, textvariable = message, background = "lightyellow", width = 40)

MESSAGE.grid(row = 0, column = 1, columnspan = 3, sticky = ('w'))

# Command area

DOQUIT = tk.Button(cmdframe, text = "Quit", foreground = "red", activeforeground = "red", width = 4,
                    cursor = "hand2", command = doQuit)

DOQUIT.grid(row = 0, column = 0, sticky = ('w'))

STARTSINGLE = tk.Button(cmdframe, text = "Start Single Sync", foreground = "blue", activeforeground =
"blue", width = 12,
                        cursor = "hand2", state = "disabled", command = startSingleSync)

STARTSINGLE.grid(row = 0, column = 2, sticky = ('w'))

STARTSUB = tk.Button(cmdframe, text = "Start Sub Sync", foreground = "blue", activeforeground = "blue",
width = 12,
                  cursor = "hand2", state = "disabled", command = startSubSync)

STARTSUB.grid(row = 0, column = 3, sticky = ('w'))

# # Immediate function calls

setPwd()

clearMessage()

readFiles()

loadConfig()

# Bindings

# Global layouts

for child in mainframe.winfo_children():child.grid_configure(padx = 2, pady = 2)
for child in schedframe.winfo_children():child.grid_configure(padx = 7, pady = 2)
for child in dispframe.winfo_children():child.grid_configure(padx = 2, pady = 2)
for child in snglframe.winfo_children():child.grid_configure(padx = 2, pady = 2)
for child in subframe.winfo_children():child.grid_configure(padx = 2, pady = 2)

```



```
for child in diskframe.wininfo_children():child.grid_configure(padx = 2, pady = 2)
for child in msgframe.wininfo_children():child.grid_configure(padx = 2, pady = 2)
for child in cmdframe.wininfo_children():child.grid_configure(padx = 2, pady = 2)
```

```
sync.mainloop()
```

```
#####
#   End of Program Code                                     #
#####
```

```
#####
#   Testing Code and Script Names, or Function Tests       #
#####
```

```
#####
#   End of Test Code ans Scripts                           #
#####
```