

## Contents

- I. Scripts Overview
- II. Script Usage

### I. Scripts Overview

Script/Folder Name	Description
TesterMonitoringTool.py	Part of the monitoring module. Maintains locally saved board profiles which are updated when new logs are detected. Should be run per tester.
SummarizerTool.py	Part of the monitoring module. Reads locally saved board profiles and outputs a summarized .CSV file. Should be run per tester.
MergedExtractTool.py	SummarizerTool functionality but instead reads directly from the logs and not from saved board profiles. Outputs a merged (diag and cal) summarized .CSV file. Can be used to process multiple testers simultaneously.
ComplianceChecker.py	Reads an input .CSV summary file and checks tester compliance. Outputs a compliance certificate.
Strike3.py	Reads an input .CSV summary file and checks Strike 3. Prints the results to the terminal and outputs a .XLSX file containing detailed failures of each board.
MTBF.py	Incomplete. Calculates number of failures within a input timeframe that will be used to calculate MTBF.
timing.py	Utility module used by all scripts to print script elapsed time to the terminal.
./config/	Contains configuration files used by the scripts.
./profiles/	Contains saved board profiles.
./output/	Folder destination for all output files.

### II. Script Usage

#### Monitoring Module (TesterMonitoringTool and Summarizer Tool)

These two scripts are used in conjunction to produce summarized logs. The ideal use case for this module is that each tester runs the monitoring tool, and summary files are generated “on-demand” by running the summarizer tool. Local board profiles are saved locally, which means generating summary files are significantly faster since they are no longer read directly from the logs (reading from the logs consists of multiple loops).

Sample use case: TesterMonitoringTool can be scheduled to run once per day (not implemented, can be done manually by using schedulers) to update the board profiles. During investigations, SummarizerTool

is used to generate a “snapshot” of the tester in question by reading from the profiles and produce a summarized .CSV file.

### TesterMonitoringTool

Argument	Description
-s or --setup	OPTIONAL. When indicated, clears existing board profiles and generates new, empty ones.
-d or --dir	REQUIRED. Directory path to monitor for new logs.

Sample usage for setup: *python TesterMonitoringTool.py -s -d ".\logfolder"*

Sample usage for monitoring “logfolder” directory: *python TesterMonitoringTool.py -d ".\logfolder"*

### SummarizerTool

Argument	Description
-m or --merge	OPTIONAL. When indicated, merges cal and diag entries in a single output. If not, diag and cal are separated.
-f or --file	OPTIONAL. Summarizes the specified .LOG file instead.






Sample usage for single file mode:

*python SummarizerTool.py -f ".\logfolder\Cal\_CLPREA100\_03302020.log"*

Sample usage for normal mode with merged summary: *python SummarizerTool.py -m*

### MergedExtractTool

The ideal use case for this script is when all the log files are saved in a server and there is a need to produce the summaries in bulk. The input folder must contain separated tester folders with each tester folder containing the diag and cal logs. For example, an input folder named “samplelogs” could contain the following folders:

 clprea83	7/29/2020 4:11 PM	File folder
 clprea85	8/12/2020 1:19 PM	File folder
 clprea90	8/12/2020 1:19 PM	File folder
 clprea93	7/29/2020 4:11 PM	File folder
 clprea94	7/29/2020 4:11 PM	File folder
 clprea95	8/12/2020 1:19 PM	File folder
 clprea100	7/29/2020 4:11 PM	File folder
 clprea102	7/29/2020 4:11 PM	File folder
 clprea106	8/12/2020 1:49 PM	File folder

Inside each of these are the diag and cal logs.

Argument	Description
-d or --dir	REQUIRED. Directory containing the tester folders and log files.

Sample usage to read from "samplelogs" located in Desktop:

*python MergedExtractTool.py -d "C:\Users\user\Desktop\samplelogs"*

### ComplianceChecker

Reads all summarized .CSV files in /output/ and checks the compliance of each tester.

Argument	Description
-n or --name	REQUIRED. Employee Name
-i or --id	REQUIRED. Employee ID

Sample usage: *python ComplianceChecker.py -n Zildjian Restituto -i a0489136*

### Strike3

Produces a Strike 3 report of a tester based from the input .CSV file. Only processes entries that are within the input date range.

Argument	Description
-f or --file	REQUIRED. Location path of the input .CSV file
-s or --start	REQUIRED. Starting date of Strike 3 check
-e or --end	REQUIRED. End date of Strike 3 check

Sample usage to check Strike 3 of EA95 from 1/1/2020 to 5/2/2020 :

*python .\Strike3.py -f "C:\Users\user\Desktop\CLPREA95\_Summary\_Cal\_Diag\_All\_08132020.csv" -s 1/1/2020 -e 5/2/2020*

### MTBF

Counts the number of fails within the input date range.

Argument	Description
-f or --file	REQUIRED. Location path of the input .CSV file
-s or --start	REQUIRED. Starting date of Strike 3 check
-e or --end	REQUIRED. End date of Strike 3 check

Sample usage to check number of fails of EA95 from 1/1/2020 to 5/2/2020 :

```
python.\MTBF.py -f "C:\Users\user\Desktop\CLPREA95_Summary_Cal_Diag_All_08132020.csv" -s  
1/1/2020 -e 5/2/2020
```

## Config Folder

### **rx\_cal**

This file contains regex unique to cal logs. To add a new board, simply add a new line, with the unique identifier and the regex separated by a comma. The last three lines (*dut*, *self\_test*, *backup\_date*) are not board definitions. The *dut* line is used to detect DIB model and part number. The *self\_test* line is to detect self-test results. Finally, the *backup\_date* line is used to capture test date/time in case self-test fails.

### **rx\_cal\_res**

This file contains regex unique to cal logs. The definitions are used to capture calibration test results and their respective date and time.

### **rx\_diag**

This file contains regex unique to diag logs. The *dut* line is used to detect DIB model. The *diag\_table* lines are used to parse the summary tables in the diag logs. The *testing* line is used to parse individual test lines to capture board serial numbers. The *error* line is used to capture error codes.

### **setup.log**

This file contains a snippet of a sample diag file to capture expected board types. UNTESTED: If there are undetected boards in the generated summary files, you may change the contents of this file with that of the specific tester's sample diags (must be a full diag run to include all boards).