

**xPlorer RWR Java Plugin: Configuration file**

|  |  |
| --- | --- |
| Author(s): | John Kang |
| Creation Date: | July 18, 2019 |

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **By** | **Comment** |
| 07/18/2019 | 0.1 | John Kang | Initial Draft. |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |



Revision History 2

Reference 2

List of Figures 4

List of Tables 4

Reference Documents 4

List of Acronyms 5

1 Overview 6

2 REQUIREMENTS 6

2.1 Setup 6

2.2 Time Requirement 6

2.3 Space Requirement 6

3 Changes Impact 6

4 Methodology 7

5 Logic 8

5.1 Psuedocode 10

5.2 Design Flow 11

5.3 Init() 11

5.4 Execute() 11

5.5 Destroy() 14

6 Limitations 15

7 Future 15

List of Figures

Figure 1 Design Flow of the code 10

Figure 2 Index matching of name and parameter of the SET event 11





List of Acronyms

|  |  |
| --- | --- |
| Acronym | Definition |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

# Documentation: ConfigurationFileRWRPlugin

**Overview**

Look at performance of the drive and see where the drive lowered in performance. Plugin helps link SET events together so that engineers can debug faster rather than manually trace through the xPlorer GUI

Ex. Showing time delta from start and end of command

Ex. Showing how many times a specific SET event occurred in a command

**Requirements**

* xPlorer version: Standalone 2.9.5\_b43
  + <https://artifactory.wdc.com/artifactory/xplorer-tool-global/integration/bronze/2.9.5_b43/>
  + File Directories needed:
    1. RWR files
    2. Plugins
    3. xTools/app/decoder (also where resulting csv will go)
* Informer board
* Calypso-X
* Using Gen4 AMD motherboard/cpu as host PC
* Diskspd – Microsoft performance test **\*add link\***

Time Complexity:

* If no logic or minimal logic (including no large data structures with longer than O(log(n)) access time) is performed in the Plugin execute() method, each 100mb rwr file (6 million SET events) takes 10 seconds to process.
  + Ex. 200 rwr files (20gb) would take **30 minutes**
  + Minimum runtime = **# of RWR files (size 100mb)** x 10 seconds
  + 10 seconds to run through a 100,000 kb file (100 mb, .1gb)
  + One 100mb RWR file has 6 million SET events

Space Complexity:

* EXPLAINED LATER IN CONFIG FILE EXPLANATION

**Changes Impact**

* Manual tracing of the SET events should be alleviated
* More robust filter than the xPlorer GUI one

**Methodology**

1. Load the correct production with the SETevents you want to capture onto the drive
2. Create and Run xPlorer session (stop when last RWR is loaded/test stops
3. Run diskspd
   1. Ex. diskspd64.exe %SEQUENTIAL%%READ%%b128KiB%%QD32%%Thread1%%NoSWCache%%TestLatancy%%WarmUpTime%%TEST\_DURATION%%TLC\_BIN% > %LogFolder%\TLC\_128KiB\_SeqRead\_QD32.log

@set LogFolder=C:\Diskspd\_Log\

@IF NOT EXIST %LogFolder% mkdir %LogFolder%

@REM Test Setting

@set TEST\_DURATION=-d60

@set WarmUpTime=-W5

@REM Test Parameter

@set NoSWCache=-S

@set TestLatancy=-L

@REM File Location

@set XBFS\_PARTITION=R:\

@set SLC\_PARTITION=S:\

@set TLC\_PARTITION=T:\

@set XBFS\_BIN=%XBFS\_PARTITION%XBFS.bin

@set SLC\_BIN=%SLC\_PARTITION%SLC.bin

@set TLC\_BIN=%TLC\_PARTITION%TLC.bin

@REM Test Type

@set RANDOM=-r4k

@set SEQUENTIAL=

@set READ=-w0

@set WRITE=-w100

@REM Block Size

@set b4KiB=-b4K

@set b32KiB=-b32K

@set b128KiB=-b128K

@REM Queue Depth & Thread

@set QD1=-o1

@set QD32=-o32

@set QD64=-o64

@set QD128=-o128

@set Thread1=-t1

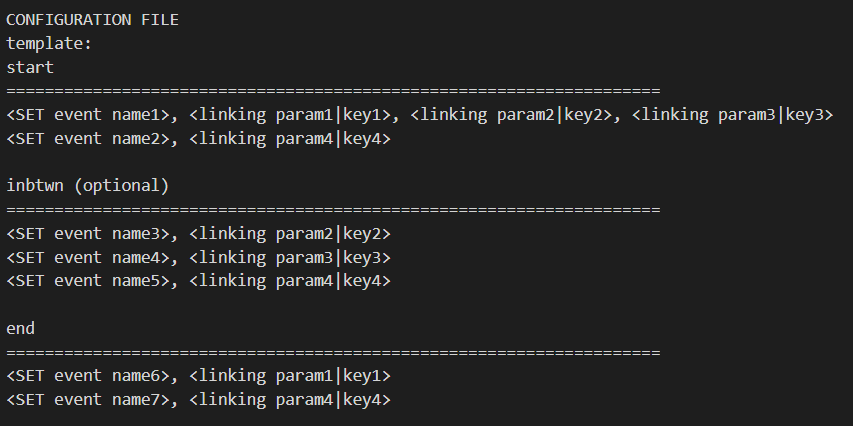
1. Capture RWR files through the informer
2. Run **“xrwrdecoder-cli.bat process decode --product=Calypso --rwr-path<path where RWR files are> --dco-path=C:\RWR\SetDictionary.dco --output-type=NONE --plugin-classes=<name of plugin class> --plugin-source-directory=<where the folder the plugin is>”** in the file path **xtools/app/xrwrdecoder**
3. Output into CSV file

**Logic**

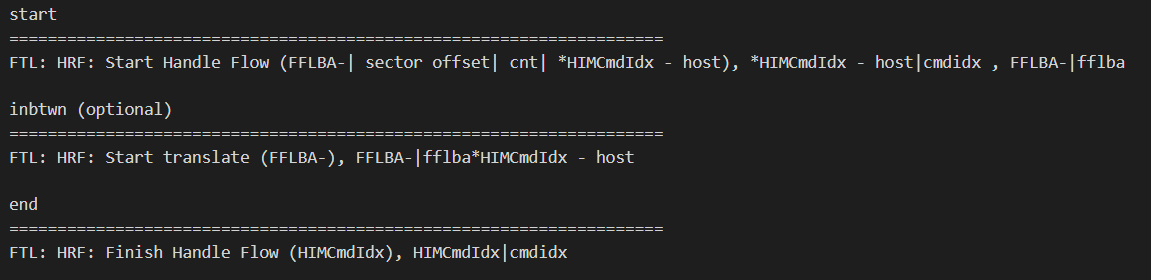
The xPlorer plugin has 3 stages:

1. Init
2. Execute
3. Destroy

This plugin requires the input of a configuration file (simple .txt file)

Template 

Example



Data structures needed:

***start\_map***: <key: set name, value: list[linking\_parameter]>

***inbtwn\_map***: <key: set name, value: linking\_parameter>

***end\_map***: <key: set name, value: linking\_parameter>

***map\_of\_linking\_parameter\_to\_key***: <key: linking\_parameter, value: key>

***map\_of\_key+hexvalue\_to\_track***: <key+0x000, value: track#>

***map\_of\_track\_and\_events***: <key: track#, value: list[set events]>

int ***track#*** = 0;

int ***maxResults*** = 0;

int ***currentResult*** = 0;

***List<Long> topResults*** = new ArrayList<>();

*\*Bracket <> and , notations in the configuration template are needed for init stage to parse the Strings correctly.*

***Start map, in between map,*** and ***end map***: maps that signify to the plugin that these SET events are important.

***Start map*** has events that indicate the start of the trace.

***Inbtwn map*** is everything you deem important in between the start.

***End map*** has events that indicate the stop of the trace, collected all the information you needed for the trace.

* + - The keyis the SET event parameter name and the value is the exact syntax of the linking parameter you want to use

**Ex.** ***KEY -*** FTL: HRF: Start Handle Flow (FFLBA-| sector offset| cnt| \*HIMCmdIdx - host) ***VALUE -*** \*HIMCmdIdx – host or FFLBA-

*\*Ending map set events MUST have a keyword that matches with a start events keyword in order for the trace to end.*

***Map\_of\_linking\_paramter\_to\_key***: used to pair the linking parameter with a similar keyword, this is needed because sometimes parameters aren’t spelt the same **ex. \*HIMCmdIdx – host** and **HIMCmdIdx**

***Map\_of\_track\_and\_events***: used to group SET events related to one trace together. Dumped when trace is complete, meaning start and end was found so memory/RAM usage doesn’t get too high.

***Map\_of\_key+hexvalue\_to\_track***: used when you find a matching keyword+hexvalue in the map, you can link the SET event to the original trace because multiple traces can occur at the same time before closing, not sequential. We use a keyword + hexvalue because there could be an edge case where certain hexvalue equal each other like 0x000 fflba and a 0x000 vba and a 0x000 command index.

Because this is a Java Class plugin there is no main() method which means the next few data members of the class act as “globals” so that each method: init(), execute(), and destroy(), can use them.

* Int ***track#***, initialized outside of the execute method so it doesn’t reset all the time Track number should be **unique** to each time a start event occurs.
* Int ***maxResult***, set in the init by the configuration, indicated how many of the outliers you want
* Int ***currentResult***, used as a **counter** up to maxResult to populate the topResults Arraylist with the first N

***List<Long> topResults*** should only contain the **top N number** of results you wanted

Ex. If you want 20 of the longest commands then the array should only have the top 20

\*the results will be dumped to the csv file in the destroy() method.

***Psuedocode***:

//read config file (init)

if start

put name of SETevent into a starter\_map with linking\_parameter

put linking\_parameter into map\_of\_linking\_parameter\_to\_key with key

if end

put name of SETevent into a ender\_map with linking\_parameter

put linking\_parameter into map\_of\_linking\_parameter\_to\_key with key

if inbtwn

put name of SETevent into a ender\_map with linking\_parameter

put linking\_parameter into map\_of\_linking\_parameter\_to\_key with key

//parsing (execute)

foreach row in file //done for us by the plugin

if SETevent is in starter\_map

for(linking\_param in list[linking\_param])

1. find the linking\_param in the name and then the actual hex value in parameter

2. find key by map\_of\_parameter\_key[linking\_param]

keyhex = 1 + 2;

3. put "key + hexadecimal" in map\_of\_hexvalue\_to\_track with track#

4. map[track#] add list[[timestamp, SETevent name, linking\_param]]

5. increment track# ++;

if SETevent is in ender\_map

1. find the key by using map\_of\_linking\_param\_to\_key[linking\_param], linking\_param can be found by ender\_map[SETevent]

2. find the linking\_param in the name and then the actual hex value in the parameter

keyhex = 1 + 2;

3. if "key + hexadecimal" in map\_of\_hexvalue\_to\_track

track# = map\_of\_hexvalue\_to\_track[key + hexadecimal]

map\_of\_track\_and\_events[track#] add list[timestamp, SETevent]

//calculate

- simple: take the time delta from start\_event to end\_event

- DUMP list from map\_of\_track\_and\_events[track#] to csv file //add trigger if you want setevents or not

- DUMP "key + hexadecimal" and time delta to CSV.

- //remove track from dictionary to save space,

3. else do nothing

if SETevent is in inbtwn\_map

1. find the key by using map\_of\_linking\_param\_to\_key[linking\_param], linking\_param can be found by ender\_map[SETevent]

2. find the linking\_param in the name and then the actual hex value in the parameter

keyhex = 1 + 2;

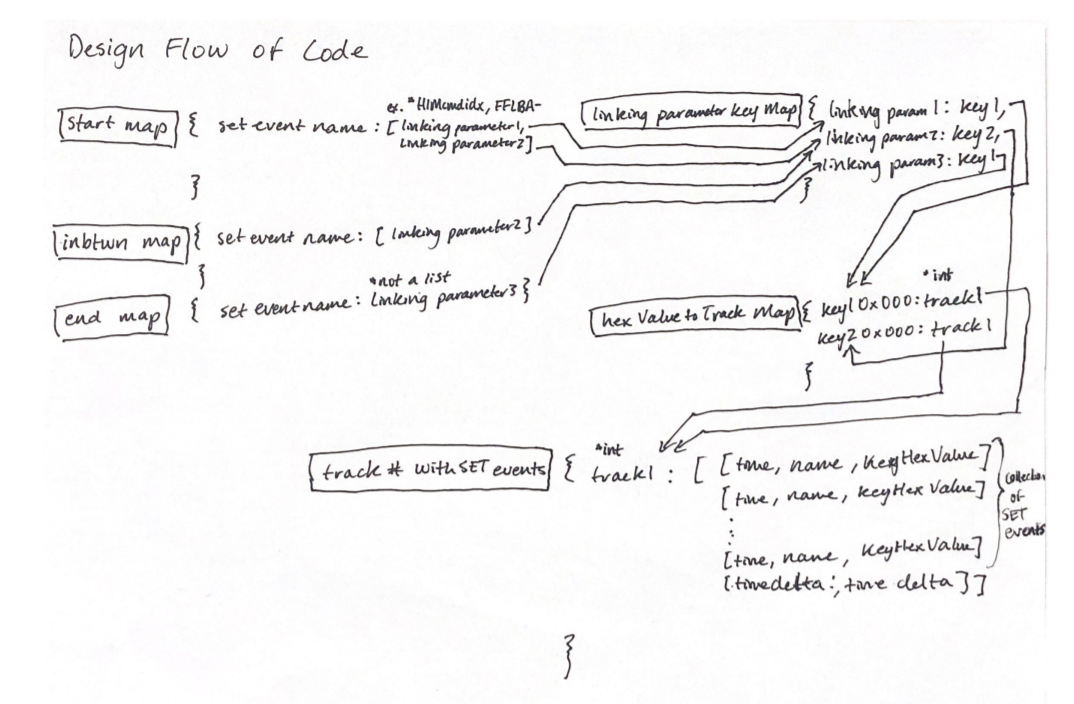
3. if "key + hexadecimal" in map\_of\_hexvalue\_to\_track

track# = map\_of\_hexvalue\_to\_track[key + hexadecimal]

map\_of\_track\_and\_events[track#] add list[timestamp, SETevent]

//dump (destroy)

Dump topResults data structure into the csv file.



*Figure 1 Design Flow of Code*

**INIT**

try {  
 FileReader fr = new FileReader("C:\\xTools\\app\\xrwrdecoder\\configfile.txt");  
  
 int i;  
 while ((i = fr.read()) != -1)  
 System.*out*.print((char) i);  
} catch (FileNotFoundException e){

This is the init phase, simple ***file reader*** that will manipulate the strings to the proper style and then place them into the appropriate data structures:

* start\_map: <key: set name, value: list[linking\_parameter]>
* inbtwn\_map: <key: set name, value: linking\_parameter>
* end\_map: <key: set name, value: linking\_parameter>
* map\_of\_linking\_parameter\_to\_key: <key: linking\_parameter, value: key>
* int maxResults = 0;

**EXECUTE**

START LOGIC

if(startMap.containsKey(name)){  
 for(String linkingParameter : startMap.get(name)) {

ex. *start\_map: {*

*FTL: HRF: Start Handle Flow (FFLBA-| sector offset| cnt| \*HIMCmdIdx - host): [\*HIMCmdIdx - host, FFLBA-]*

*}*

First if statement is to check if the name of the SET event indicates a starting event (start of a trace).

Iterate through the list of linking parameter it has, these are associated with the keywords we want to trace.

String key = mapOfLinkingParameterToKey.get(linkingParameter);  
String lowerLinkingParameter = linkingParameter.toLowerCase();  
String[] arrayOfParameters = (name.substring(name.indexOf("(") + 1, name.indexOf(")"))).toLowerCase().split("\\|");  
String[] arrayOfHexValues = parameters.split("\\|");

*map\_of\_linking\_param\_to\_key: {*

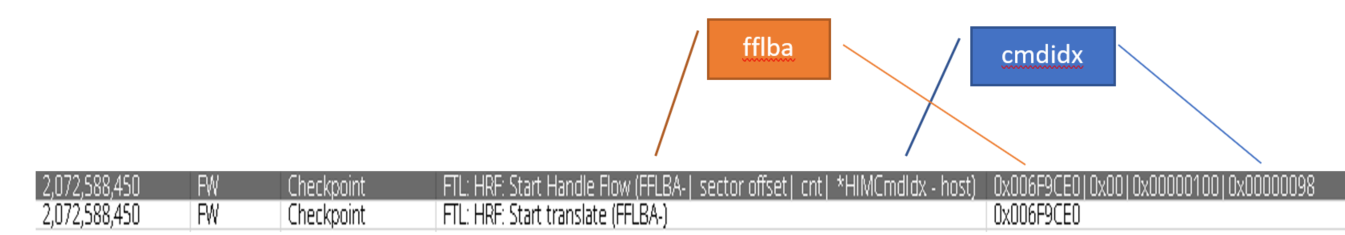
*\*HIMCmdIdx - host: cmdidx,*

*HIMCmdIdx: cmdidx,*

*FFLBA-: fflba*

*}*

Keyword is found by matching the parameter name:



*Figure 2 Index matching of name and parameter*

Next we want to **break the parameter names in the name into an array** so we can match it to the actual hex on the other side

for(int i = 0; i < arrayOfParameters.length; i++){  
 if(arrayOfParameters[i].contains(lowerLinkingParameter)){  
 indexOfLink = i;  
 break;  
 }  
}

this loop looks for the parameter name so we can use the same index on the other side

String hexValue = arrayOfHexValues[indexOfLink];

String keyAndHex = key + hexValue;  
mapOfHexValueToTrackNumber.put(keyAndHex, trackNumber);  
mapOfTrackAndSETevents.put(trackNumber, new ArrayList<String[]>());  
String[] usefulInfo = new String[]{timestamp.toString(), name, keyAndHex};  
mapOfTrackAndSETevents.get(trackNumber).add(usefulInfo);

***KeyAndHex*** combines the key and the actual hex value to avoid an edge case where certain hexvalue equal each other

*like 0x000 fflba and a 0x000 vba and a 0x000 command index.*

This is also placed in a ***mapOfHexValueToTrackNumber*** so when we find it later we can find the data structure that has the related SET events to that command/address/etc. in this case either cmd idx or fflba.

trackNumber++;

finally, the track number is increased because multiple starts can occur before seeing an end, not sequential, increasing the track number **avoids any collisions**.

END LOGIC

if(endMap.containsKey(name)) {  
 //System.out.println(name + " " + endcounter);  
 endcounter++;  
 String linkingParameter = endMap.get(name);  
 String key = mapOfLinkingParameterToKey.get(linkingParameter);  
 String lowerLinkingParameter = linkingParameter.toLowerCase();  
 String[] arrayOfParameters = (name.substring(name.indexOf("(") + 1, name.indexOf(")"))).toLowerCase().split("\\|");  
 String[] arrayOfHexValues = parameters.split("\\|");  
 int indexOfLink = 0;  
 for (int i = 0; i < arrayOfParameters.length; i++) {  
 if (arrayOfParameters[i].contains(lowerLinkingParameter)) {  
 indexOfLink = i;  
 break;  
 }  
 }  
 String hexValue = arrayOfHexValues[indexOfLink];  
 String keyAndHex = key + hexValue;

this is like the START where we need to find the keyword from the ***linking\_param\_to\_key map*** and then finding the actual hex value by making the String an array and then indexing where the linking parameter name is.

if (mapOfHexValueToTrackNumber.containsKey(keyAndHex)) {  
 int track = mapOfHexValueToTrackNumber.get(keyAndHex);  
  
 Long timeDelta = timestamp - Long.*parseLong*(mapOfTrackAndSETevents.get(track).get(0)[0]);

next is to check if the ***keyAndHex*** is in the ***mapOfHexValueToTrackNumber***, this will see if there was ever a start to the trace, there are times where you only check only a section of the RWR files and could have ending events but no start, we should ignore those.

If it is there the track number can be found in the ***mapOfHexValueToTrackNumber***

Then we can get the time delta by using the first entry of the trace which is the start which will also have the timestamp.

if(currentResult < 20){  
 topResults.add(timeDelta);  
 Collections.*sort*(topResults, Collections.*reverseOrder*());  
 currentResult++;  
} else {  
 if(timeDelta > topResults.get(maxResults - 1)){  
 topResults.set(maxResults - 1, timeDelta);  
 Collections.*sort*(topResults, Collections.*reverseOrder*());  
  
 //adding setevents to track number  
 String[] usefulInfo = new String[]{timestamp.toString(), name, keyAndHex};  
 mapOfTrackAndSETevents.get(track).add(usefulInfo);  
  
 String[] calculatedInfo = new String[]{"Time Delta", timeDelta.toString()};  
 mapOfTrackAndSETevents.get(track).add(calculatedInfo);  
  
 for (String[] rowData : mapOfTrackAndSETevents.get(track)){  
 *fileWriter*.append(String.*join*(",", rowData));  
 *fileWriter*.append("\n");  
 }  
 }  
}

Next, we need to check if the result is the first 20 (max results or outliers we want from the RWR files) because we need to **populate our list** with data the rest of the time deltas can compare with.

**List is sorted** so that we only need to check the 20th entry in the list, if its greater than we can **replace the data** with the time delta we currently have and then we need to sort it again because it might not necessarily be the 20th worst time.

Then we can **add the ending SET event** to the ***track number map*** that has been holding all our other SET events like start event or in between events.

Then we can **file write it to a CSV** file because…

mapOfTrackAndSETevents.remove(track);  
mapOfHexValueToTrackNumber.remove(keyAndHex);

we do ***NOT*** want to hold onto this data. Reason being:

* In my example of finding ***FTL: HRF: Start Handle Flow (FFLBA-| sector offset| cnt| \*HIMCmdIdx - host)*** this event occurred ***37818 times***, which means 37818 tracks were made, increasing the size of our ***map\_track\_to\_setevents*** data structure. To alleviate this strain on the memory we can dump as the finish the track.
* We want to prevent hashmap **look up time** increasing and reduce memory usage.

**DESTROY**

try{  
 for (Long results : topResults) {  
 *fileWriter*.append(results.toString());  
 *fileWriter*.append("\n");  
 }  
 *fileWriter*.flush();  
 *fileWriter*.close();  
} catch (IOException e){

This executes when it’s done going through all the RWR files and we want to ***display all the top results*** we got into the CSV file and then close it as well.

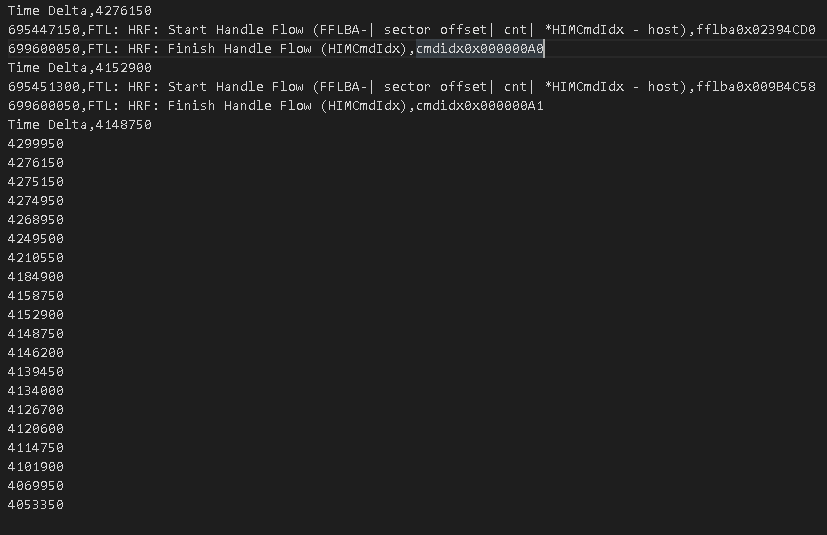
**Limitations**

**1.**

Events like this don’t have a parameter name in the name, logic to link parameter name to actual hex value wouldn’t work.

**2.**

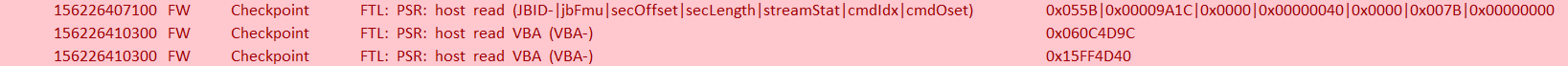
Even though at the end of the csv file has the 20 (or however many worst cases you want) the csv contains a lot more than 20 SET event traces, this is due to in the END logic of the EXECUTE method, we write to the csv file whenever the time delta beats the CURRENT worst 20 times.



**Future**

Adding logic that add events that are sequential but don’t have a linking parameter.

**Ex.**



Adding logic that just counts the number of specific events that occurred within a command

**Ex.**

How many times did this happen within the command?

