

**xPlorer RWR Java Plugin: Configuration file**

**(Command Execution Worse Outliers Finder)**

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

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **By** | **Comment** |
| 07/18/2019 | 1.0 | John Kang | Initial Draft. |
| 08/08/2019 | 1.1 | John Kang | Making Nicer |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |



Revision History 2

Reference 2

Table of Content 3

List of Figures 4

List of Tables 4

Reference Documents 4

List of Acronyms 5

1 Overview 6

2 Metholody 7

3 Logic 8

4 Future 11

5 Detailed Explanation 12

5.1 Psuedocode 10

5.2 Design Flow 11

5.3 Init() 14

5.4 Execute() 14

5.5 Destroy() 16

6 Additional Features 17

7 Limitations 18

8 Future 18

List of Figures

Figure 1 Design Flow of the code 10

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



Empty\*



List of Acronyms

|  |  |
| --- | --- |
| Acronym | Definition |
| Inbtwn | In between |
| Track | Collection of related SET events |
|  |  |
|  |  |
|  |  |
|  |  |

# Documentation: ConfigurationFileRWRPlugin

**Overview**

The “Configuration File RWR” plugin extends the Explorer functionality to add a powerful capability to “link” SET events that are generated during the execution of a test. Once linked, the plugin can filter or extract information from the SET events automatically, freeing the engineer from doing this work manually.

For example, the plugin can:

* Calculate the overall time taken to execute a host command and generate a histogram of the resultant times
* Dump the events associated with commands with the longest execution times or that exceeded a specified threshold.
* Perform logic on linked SET events and dump the events corresponding to user-specified criteria such as specific addresses, or number of specified operations, or failure cases.
* Etc…

The plugin can operate both while the test is running, parsing the SET event traces in real-time (and therefore avoiding the need to generate GBs of rwr files), or can re-parse existing rwr files.

All the plugin logic is controlled by a human-readable and editable text file where the SET events of interest are specified, along with the logic for how the plugin should associated SET events.

**Methodology**

*Steps to run the plugin and xPlorer CLI*

xPlorer CLI-Plugin

1. Find file path of where Plugin is.
2. Edit configuration file as needed for what you need. (need to file path of this too)

If running test live with plugin:

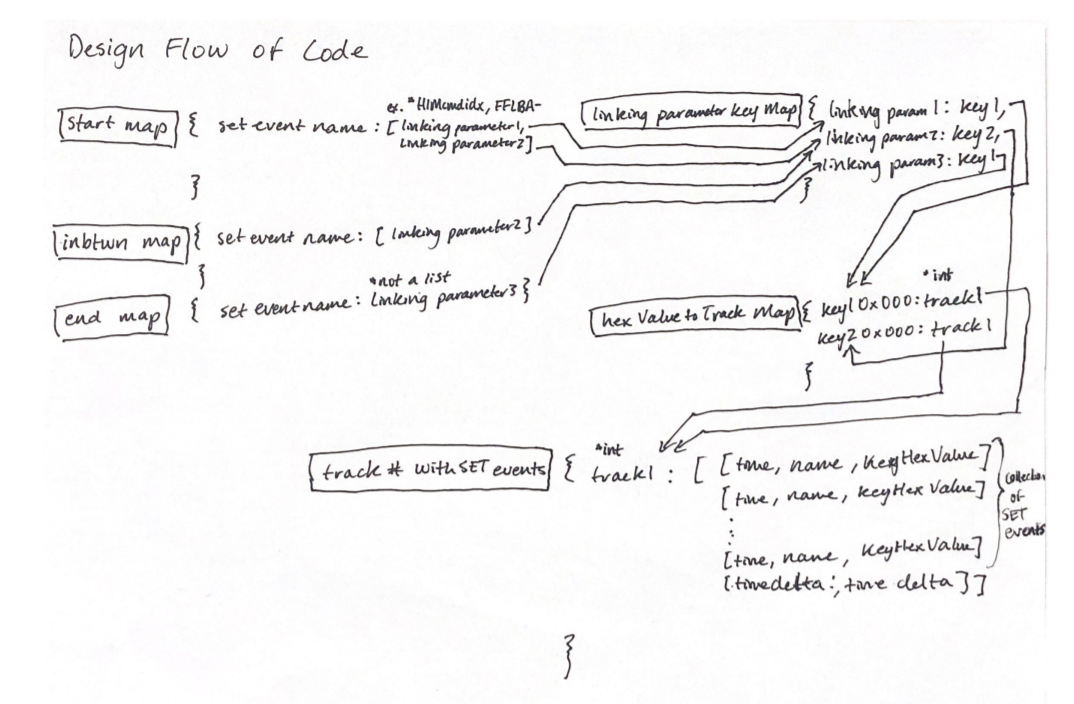
1. **File Path to CMD:** cd C:\xTools\app\xplorer\
2. **START:** xplorer-cli.bat session create --session-configuration-file=<path to json configuration file> --start=true
3. **STOP:** xplorer-cli.bat session stop --session-id=<session-id>
4. “Results.csv” will output into xrwrdecoder directory in xTools (plugin will do this automatically)

If RWR events where already captured:

1. Find file path of where RWR files are

1. Run **“xrwrdecoder-cli.bat process decode --product=<product name> --run-mode=READ --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>”** command in the file path **xtools/app/xrwrdecoder**
2. Results will output into CSV file (plugin will do this automatically)

**Logic**



*Figure 1 Design Flow of Code*

**PROGRAMMING LANGUAGE DISTINCTIONS**

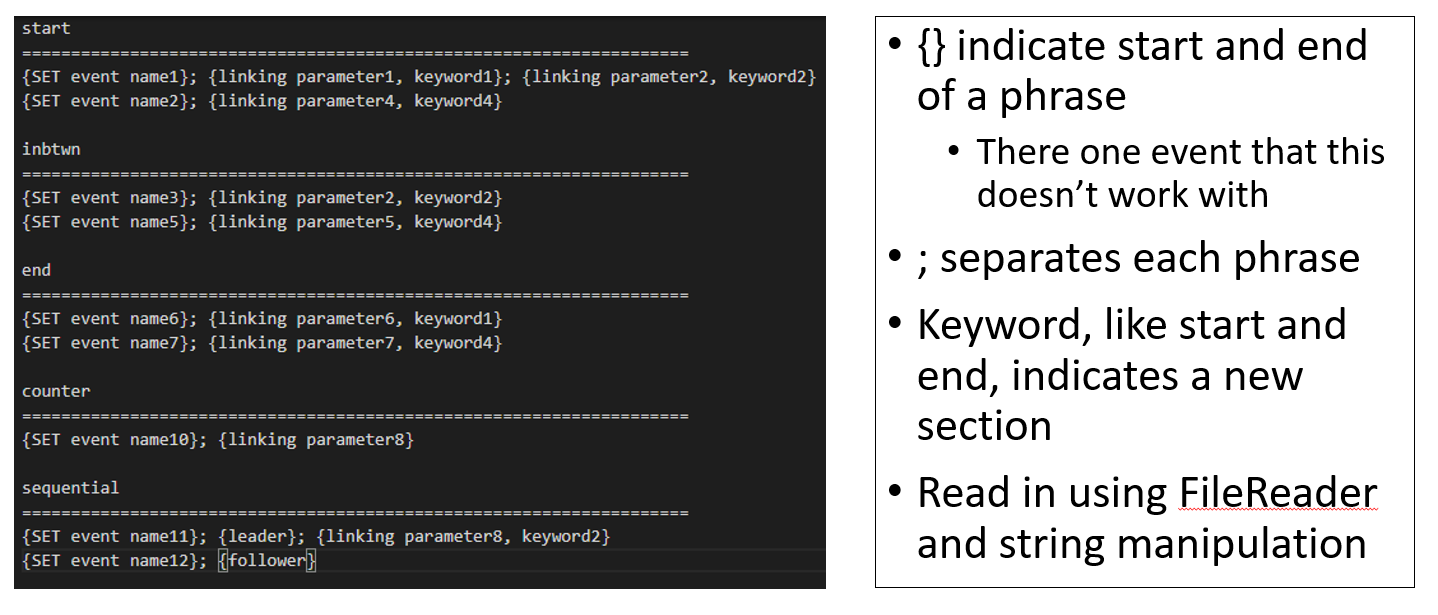
C++/Java use the term “Map/HashMap” for their data structure of key value pairs.

Python uses the term “Dictionary” for their data structure of key value pairs.

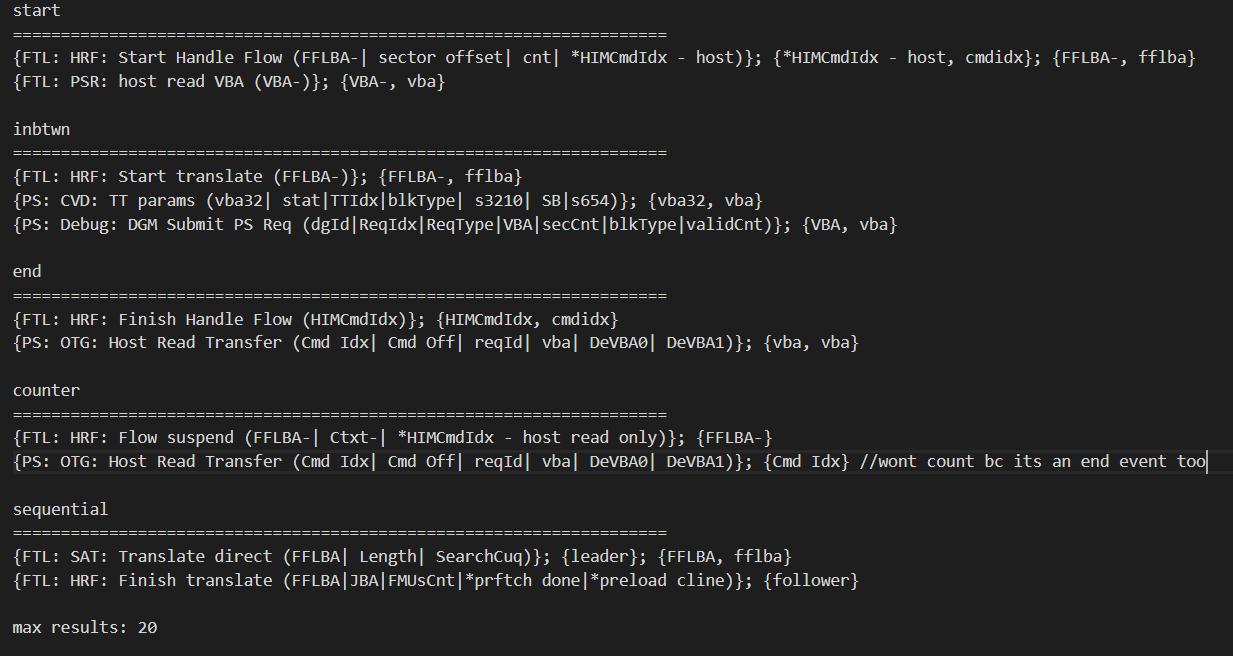
**A map is the most prominent data structure used in the design flow, is it used to “map” a unique key with a value.**

**The key can be any data type, and the value can be any data type but also can be any data structure, in this design there can be a list as the value for each specific key.**

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

Template 

Example



Higher Level understanding of Design flow:

In the configuration file SET events are sectioned off into different “indicators”

1. There are events that start the trace “starters”.
2. There are events you want to track alongside with the start “in between”.
3. There are events that you need to stop the trace “enders” these signify the end of a trace

All these events’ names are placed in separate maps (start, inbtwn, and end) for easy access time when looking them up

\*these events are linked by the keyword.

Along with the maps mention there is **a map that links actual parameter name syntax with a keyword** like “\*HIMCmdIdx – host” and “HIMCmdIdx” have keywords “cmd idx” (set by the user) so that the program knows, that even if the syntax is off, the keywords are the same and can link the two events.

The **structure** of the configuration file is:

1. Each phrase will be bracket by {}
2. The **first phrase** is the key (SET event name – must match name exactly) of each respective map (start, inbtwn, end)
3. Each specific parameter name, after the SET event name, you want for that specific SET event like “FFLBA-“ is also bracketed by {} with a keyword like fflba,
   1. first element is the key and the second element is the value in the map, delimitated by a “,”. Ex. {FFLBA-, fflba}

Edge cases that will not work as intended:

1. HW events don’t have parameters in the name
2. Some SET events don’t have () to separate the parameters from the name, this will break if the first element is the linking parameter you want to use. ***Workaround***: use the name along with linking parameter.
3. The plugin reads the parameter hex values as strings so if the length of the hex value doesn’t match then the plugin will think they’re different. ***Example***: cmd idx 0x0049 != cmd idx 0x00000049. ***Workaround:*** (future implementation) use a string to hex number function.

Then when going through the RWR files, as each SET event comes in we check if the name of the SET event is in any of the maps to see if it’s an important event we want to place in a Track.

**Track is a terminology used to describe a collection of SET events related to each other through their parameters.**

Then depending on what it is a start, end, or in between event we perform different steps for each.

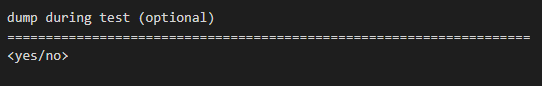
1. **Start** events will start the Track of a command putting each SET event’s time stamp, name, parameters into a list of SET events, acting like a bucket for collecting events relative to the start event based on the linking parameter.
   1. These can run parallel as multiple start events can occur before ending, creating multiple tracks.
2. **Inbtwn** events will indicate events that the user would like to see associated with the Track and dumped into a CSV file but isn’t a start or end event.
3. **End** events, are events stored in the ender map that will allow the plugin to know when logic should be performed (like time deltas) and when to dump the Track of events from that single command trace.
   1. Track lives until it an end event is matched and then the track will be removed along with all the SET events in that track to keep the size of the data structure low.
   2. all SET events configured in the config file in the beginning can be dumped to the csv file. Showing you an accurate chronological time of each SET event.

In the very last part of the plugin, **all remaining results** that haven’t been dumped to the csv will be dumped and the csv files will close.

**Future** *(not yet implemented)*

*Adding option to dump results at the end only, potentially don’t want the SET events as they come out*

**Ex.**



**Explanation of Data Structure and Logic**

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<>();

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

1. ***Start map*** has events that indicate the start of the trace.
2. ***Inbtwn map*** is everything you deem important in between the start.
3. ***End map*** has events that indicate the stop of the trace, collected all the information you needed for the trace.

The key in each mapis 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

*\*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.** Even if they are the same

***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.

**HOW THE MAPS ARE CONNECTED**

*Explanation of the design flow diagram above*

1. When a SET event is found in the **start map**, it will create a NEW Track number in the map ***Map\_of\_track\_and\_events***, this will contain all the associated SET events.
2. When a SET event is found in the **inbtwn map**, the plugin will find the keyword associated to the specific parameter using the ***Map\_of\_linking\_paramter\_to\_key*** , then using the keyHexValue it created it will try to find it in the ***Map\_of\_key+hexvalue\_to\_track***, if it is in there it will access the track number in that map and link it to ***Map\_of\_track\_and\_events***, and this is where the SET event information like name, time, and parameters will be placed.
3. When a SET event is found in the **end map** the plugin will find the keyword associated to the specific parameter using the ***Map\_of\_linking\_paramter\_to\_key***, then using the keyHexValue it create it will try to find it in the ***Map\_of\_key+hexvalue\_to\_track***, if it is in there it will access the track number in that map and link it to ***Map\_of\_track\_and\_events***, and this is where the SET event information like name, time, and parameters will be placed.
   1. In addition to adding the SET event this is where the logic, removing, and dumping of the Tracks to a CSV file happens.

This Java Class plugin a few data members that are accessible to each method: init(), execute(), and destroy():

* 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 but number received from the configuration file, 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. For example, if you want 20 of the longest commands then the array should only have the top 20

***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.

The xPlorer plugin has 3 stages:

1. Init
2. Execute
3. Destroy

**INIT**

FileReader fr = new FileReader("C:\\xTools\\app\\xrwrdecoder\\configfile.txt");

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]>
  + linking\_parameter is the actual syntax of the parameter name like: “HIMCmdIdx”
* 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: {*

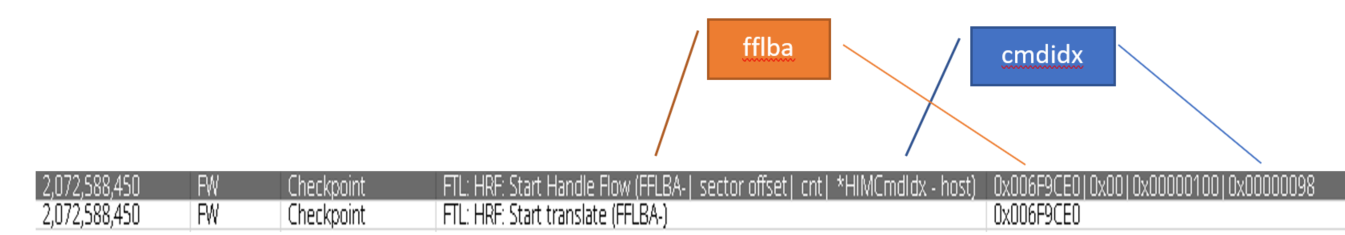
*FTL: HRF: Start Handle Flow (FFLBA-| sector offset| cnt| \*HIMCmdIdx - host): [\*HIMCmdIdx - host, FFLBA-]*

*}*

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.

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** using string manipulation in JAVA with the “SPLIT” function. So we can match it to the actual hex on the other side

String[] arrayOfHexValues = parameters.split("\\|");

***KeyAndHex*** combines the key and the actual hex value to avoid an edge case where certain hexvalue equal each other - *like (0x000 for fflba) and (0x000 for vba) and (0x000 for command index).*

This is also placed in a ***mapOfHexValueToTrackNumber*** so when we find it later we can find the track in ***mapOfTrackAndSETevents*** that has the related SET events to that command/address/etc.

***mapOfHexValueToTrackNumber*** looks like {fflba0x000: 23}

trackNumber++;

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

END LOGIC

like the START, 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)) {

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.

Track number can be found in the ***mapOfHexValueToTrackNumber***

Time delta can then be found using the timestamps stored in the ***mapOfTrackAndSETevents***

if(timeDelta > topResults.get(maxResults - 1)){  
 topResults.set(maxResults - 1, timeDelta);  
 Collections.*sort*(topResults, Collections.*reverseOrder*());

**List is sorted** so that we only need to check the Maxth 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 Maxth 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.

Each track starts at a START SET event and exists until the END SET event is met.

Deleted from *mapOfTrackAndSETevents*

Along with the track, the KeyHexValues exists only until track is deleted, there would be no use for it in the map as the track has been removed

Deleted from *mapOfHexValueToTrackNumber*

**DESTROY**

This executes when the plugin is done going through all the RWR files and dumps the final ***top results*** in the ***List<Long> topResults*** data structure into the CSV file and then close it as well.

**Additional Features**

**Counter**

* Checking how many times an event occurs within one command
* Map of event name to integer
* If an event name is encountered with the same hex value, increase counter
  + If new hex value is found add a new entry into map with counter at 1

*Psuedocode*

If counterMap contains SET event name:

Find linking parameter from counterMap

Create keyAndHex

if mapOfEventAndCounter contains keyAndHex:

increment counter;

else:

add keyAndHex to mapOfEventAndCounter and start counter at 1

**Sequential event pairing**

* Certain events while they are related don’t have parameters that the script can use to link them
* The sequential logic follows a simple pattern that the follower events related to the leader occur before a new leader event is seen
* Leaders are mapped to a group of related SET events
* Followers are mapped to leaders (then to the group)
* Leader updates when a new leader event is met. (using hex value)

*Psuedocode*

If sequentialMap contains name: //leader logic

Find linking parameter from sequentialMap

Create keyAndHex

if mapOfHexValueToTrackNumber contains keyAndHex:

Add to track

Update keyAndHex of leader in mapOfLeaderToKeyHex

else if followerToLearderMap contains name: //follower logic

get keyHex from the Leader associated to the Follower in

mapOfLeaderToKeyHex

if mapOfHexValueToTrackNumber contains keyAndHex:

Add to track

**Limitations**

**1.**

SET event names are not uniform

**2.**

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

**3.**

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.

**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:

RWR files

Plugins

xTools/app/decoder (also where output csv will go)

Informer board

Any drive with ATB/SET events enabled

Any host test system

Any test script environment: diskspd, FVT, iometer, etc.

Plugin downloaded

Configuration .txt file configured

Time Complexity:

*Based on RWR files being generated already*

Time is dependent on **3 factors**:

First, the time it takes to **complete the Test**

The test is what collects the RWR files into a directory

Second, the **number of operations** in the execute() method of the Java Plugin

This is determined by how complex the method is manipulating and accessing the data in the SET event

Third, the **size of the data structure**

No data structure should have longer than O(log(n)) access time, and because the thousands of SET events are read in seconds, data structures can grow very larger very quick, so we need to dump as SET events complete to prevent longer access/look up times.