Log Query

Introduction

When analyzing logs one of the first tasks is to search for a particular message or procedure of interest. Sometimes the point of interest in the log is just a small part of the whole log. Log Query is a searching mechanism to find and extract those parts of the log that the person analyzing the log or tester is interested in.

Simple search on a message id or a time stamp already exists in the current tools. What Log Query aims is to provide a powerful search mechanism so that entire procedures and arbitrary sequences can be searched and extracted from the log.

Log

For log query a simplified view of log is taken. A log is taken as a list of messages. Messages include actual protocol stack messages and traces.

Msg 1

Msg 2

Msg 3

…

Message

A message is a simplified view of the actual protocol stack message or trace. Only the required fields from the messages are looked upon while executing the query. This is to avoid processing the big messages, which usually contains large amount of unwanted data that the tester is not interested.

All the fields that are required for querying are extracted and kept as a (name, value) pairs. This will greatly reduce the log size as well as querying time.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | Value | Name | Value | Name | Value |

Sequence

Sequence is a range of messages in the log. Usually these are the parts of the log the tester is interested in. Usually a sequence will have a start point and an end point. Start point may be a message like RACH start and the end point may be RACH end. The start points itself can be sequences. For example: Start may be a handover to a particular cell and the end point may be going back to IDLE state.

Query

Query gives a formal mechanism to specify a sequence of interest in the log and then apply the query to interested logs to extract the sequence.

This can be better understood by comparing it with regular expressions. Regular expressions can search and extract text from strings.

Consider a string “aaabbbcccdddeeefff” and a regular expression “b\*d”. On applying the regular expression on the string we get the output as “bbbcccddd”

Imagine that the characters in the string are log messages and the whole string a complete log; we can extract sequences of messages from the log using a mechanism similar to regular expressions. Since messages are more complex than single characters a little more complex mechanism is needed.

To specify a sequence in the log we need a “**start**” marker. This is where the sequence starts. This is the only mandatory marker. Optionally we can have an “**end**” marker so that everything between the start and end will be matched.

There are two special markers: **middle** and **not**

If we specify the **middle** marker then sequences matching star and end with at least one middle is matched.

For eg: If **start** is A, **end** is D, and **middle** is C then

ACD will match

ABBCEED will match

ABBD will not match

If we specify the **not** marker then sequences matching start and end with no **not** between them.

For eg: If **start** is A, **end** is D, and **not** is C then

ABD will match

ABBCEED will not match

ABBD will match

The markers **end**, **middle**, and **not** are optional in the query.

Complex markers

The markers **start**, **end**, **middle**, and **not** can be of 3 types:

1. Match only a single message
2. Another full query ( ie marker itself can be a previously declared query )
3. Match multiple messages or queries ( ie msg1 OR msg2 OR query1 … )

Examples of single message markers are in the previous section. Here consider an example with markers as a complex query

tmpq = start : A, end : D, middle : C, not : B

Query 2 = start : G or tmpq, end : K

Then Query 2 will match the following

GK

ACDK

ACDXXXK

Query 2 will not match the following

ACBDK

ACD

Implementation and API using python

The query can be implemented in perl or python. In this document python examples are given

A query is implemented as a python dictionary or a perl hash. The keys of this dict are start, end, middle and not

Markers are implemented as a single function, or an array of functions or queries. This will be clear from the examples.

To use the query API, only one file need to be included in your script. In python the file is TagLog.py and in perl TagLog.pm

There may be slight differences in the interfaces between perl and python APIs.

The log file is converted from the armlog using another tool. The documentation for that tool is not included here. We assume that we have a log file.

To load a file use:

Log = TagLog( filename )

Consider we have a log file “log.tag”. To load the file into the script use:

Log = TagLog( “log.tag” )

A query is specified as a dict and markers as functions:

Refer the example files:

Example\_log.tag is log created for testing.

Test.py has example queries

TalLog.py had the search module.