Functional Spec

AdminQ Viewer

PSU Capstone  
Team F

Prepared for

Intel

Table of Contents

1. Introduction 3

1.1 Methodology 3

1.2 Functional Blocks 3

2. Application flowcharts 4

2.1 Overall program structure 4

2.2 Structure of BackEnd 5

2.3 Structure of FrontEnd 6

2.4 Structure of Header file 7

2.5 Structure of Map file 8

2.6 Structure of Input log files 8

3. Functionality of GUI elements 9

AdminQ Viewer Functional Spec

1. Introduction

This document describes functional specifications of the AdminQ Viewer - log parsing application of the Team F Capstone project. The specifications are derived from the meetings we had with Intel Product owner, and the research that followed in the first 3 months of the project.

The purpose of this documentation phase is to provide short specifications concerning user input, system response, system data and related hardware and logistics matters. The document will provide an overview of the classes and dataflow dependencies.

## Methodology

In the search of general approach to determine and describe the functional specifications, Team F has decided on simplified style that will introduce reader to complex ideas in methodical approach, starting with an overview following by more detailed examples.

In particular, the following principles are adopted:

* producing a comprehensible specification using diagrams and terminology which can be understood by a general audience
* writing the specifications primarily from the maintenance point of view
* avoiding unnecessary references to the technology

## Functional Blocks

The functions within the applications are described in ‘functional blocks’.

A documented functional block can consist of several levels of flowcharts with increasing level of detail. As mentioned above, finding commonality between these functional blocks is important. Where functional blocks are used in several places in the application(s) they are indicated in Overall program structure

# Application flowcharts

## Overall program structure



\*See below for more detail

* Wrapper Main calls Main methods in both FrontEnd and BackEnd
* BackEnd by default parses Header File and Code Map built in, however it will pick up new Header File and Map file residing in the same directory as the executable
* FrontEnd communicates with Backend via API layer

## Structure of BackEnd



Purpose:

Header:

* CommandFieldImpl – establishes structure of commands fields in the header file
* CommandStructImpl – establishes structure of the command struct in the header file
* ErrorImpl – establishes structure of error field in the header file
* HeaderParser – driver that utilizes above structures to parse through both the header file and the map

Logs:

* LogEntryImp -establishes a structure of a log entry
* LogParser -driver that utilizes LogEntryImp to parse through the log

## Structure of FrontEnd



Purpose:

* Controllers – collection of custom controls built to support actions on GUI panels
* CustomControls – collections of custom controls built to display fields
* Datamodels – establishes main GUI using custom data table model

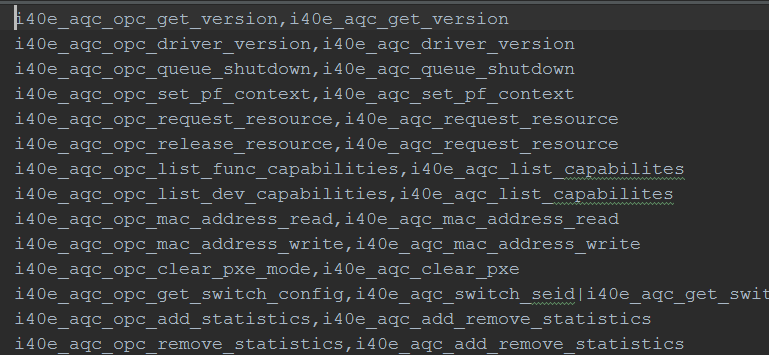
## Structure of Header file

|  |  |
| --- | --- |
| General structure of any command struct are to be followed in this pattern. The following regex collects the command structs:  struct\\s+(i40e\_aq(?:c)?\_[a-z\_]+) \\{([^}]+)}; |  |
| Op Codes are described and listed under “enum i40e\_admin\_queue\_opc”, the following regex collects op code fields:  (i40e\_aqc\_opc\_[a-z\_]+)\\s+=\\s(0x[A-F0-9]+) |  |
| Error codes are described and listed under “enum i40e\_admin\_queue\_err”. The following regex collects error codes:  (I40E\_AQ\_RC\_\\w+)\\s+=\\s([0-9]+) |  |
| Flags begin with #define and displayed in the sub-structured fashion on the right. The following regex collects flag fields:  (/\\\*(?:[\\w\\s+=;\\n])+\\\*/)|(#define\\s+([\\w\\d]+)  \\s+([\\w\\d]+))|(([\\w\\d]+)\\s+  ([\\w\\d]+)(?:\\[(\\d+)\\])?); |  |

## Structure of Map file

Map file is collection of op code objects mapped to their appropriate commands

(op codes, command struct)



## Structure of Input log files

The log files are expected to be structured in the following way

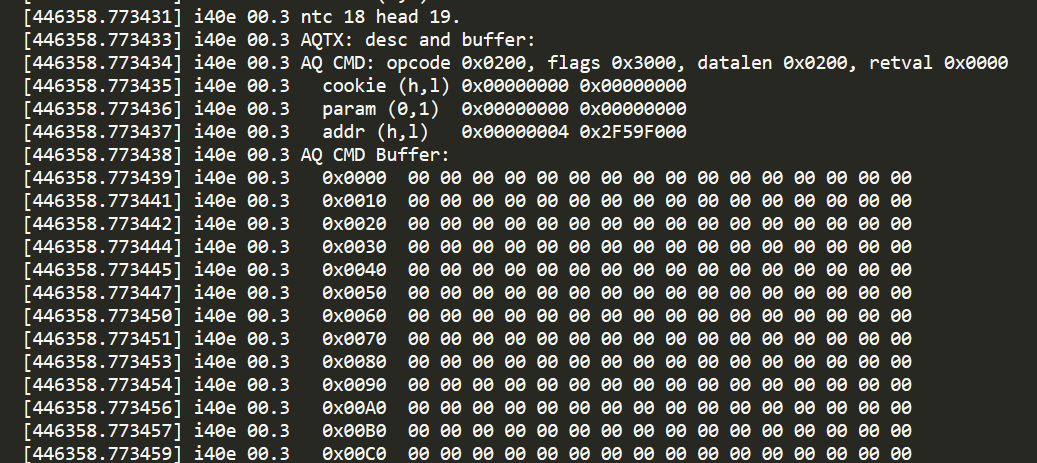
Every request/response begins with either AQTX or AQRX desc and buffer line

Device ID

Op Code Value

TimeStamp column

Flags Value



Values of mapped structure and it fields

Data buffer length

Raw Data buffer

# Functionality of GUI elements

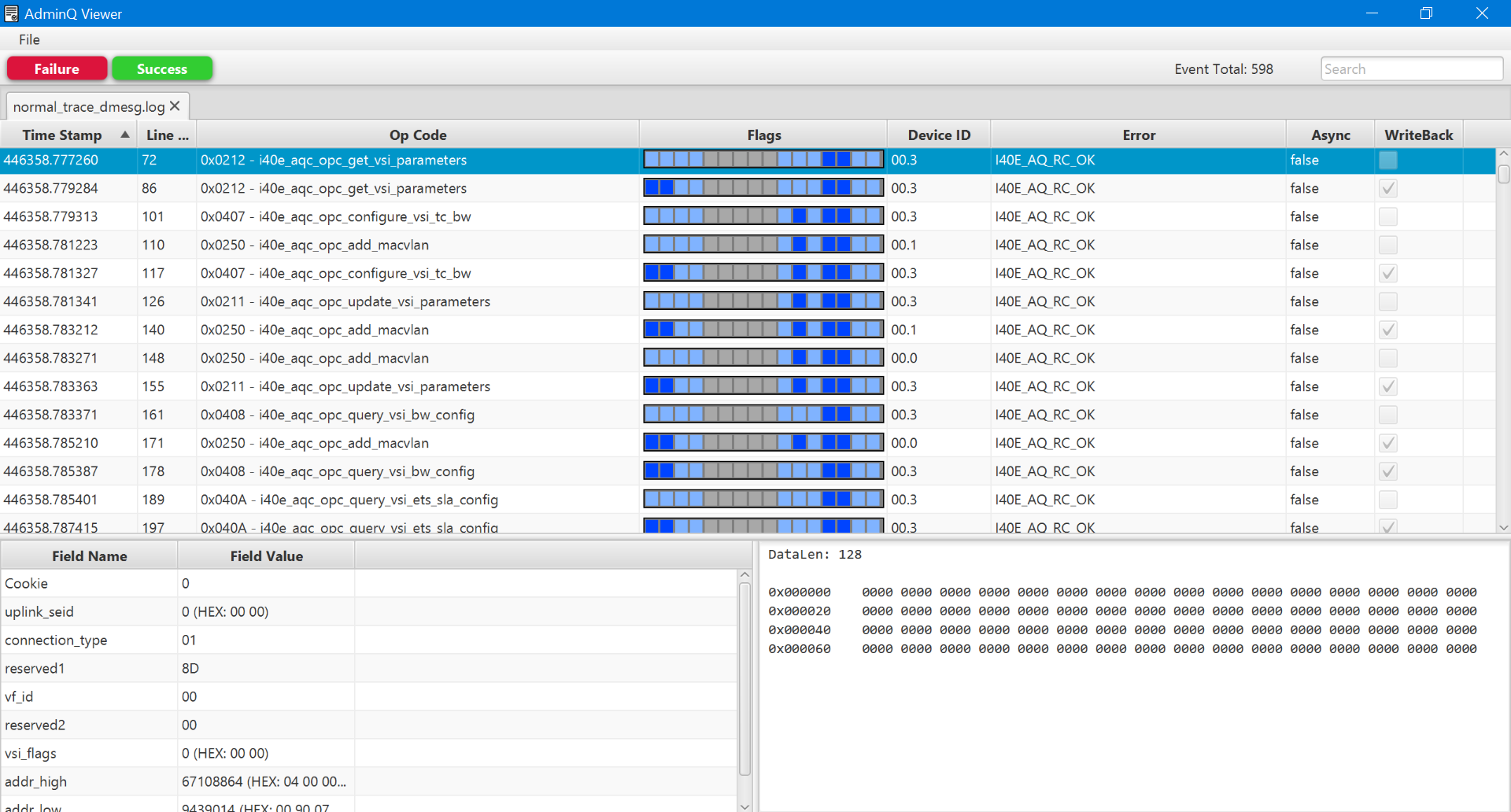
Menu for opening / drag&drop supported

Support for multiple open logs

Total statistics

CaSe sensitive search bar

Filter for successful/failed messages



Converted field values

Raw data buffer

Sortable columns

Parsed & structured data

Revision History

| Version | Date | Summary of Changes | Author | Revision Marks (Yes/No) |
| --- | --- | --- | --- | --- |
| 0.1 | 2.22.2016 | Initial revision. | Team F |  |
| 1.0 | 3.02.2016 | Final | Team F |  |
|  |  |  |  |  |