EASYLOG
A SIMPLE GUI TOOL FOR NAVIGATING LOGS ON OSX

Patricia Foster
100883992

December 15th, 2016

1. INTRODUCTION

1.1 Context

System logs are an important resource for diagnosing and troubleshooting computer problems. On Mac OSX El Capitan, the majority of system logs are stored in the */var/log* directory. Typically, log files are stored in a simple text format and can be opened and analyzed with a text editor, or with console commands such as *tail* and *grep*. Examples include system.log, wifi.log, install.log, and commerce.log. Other logs are stored as ASL – Apple System Log – files, which are binary files and require specialized software to parse and analyze.

OSX also tidies up its logs; many of the logs are 'short-term' and are deleted after a predetermined amount of time. For example, /var/log/system.log is rotated every hour using the *newsyslog* command. Other, 'long-term' logs are eventually stored in compressed gzip files to save on space, thus becoming less readily accessible.

1.2 Problem Statement

Though logs contain a wealth of important diagnostic information, this information is scattered throughout multiple files that are large and packed with extraneous details. On OSX, typical log files can run anywhere from several hundred kilobytes to several hundred megabytes. Proper searching and filtering tools are therefore essential for extracting useful intelligence.

While Mac OSX has a default GUI application to display and navigate through system logs, the search functionality is basic. Most open-source software for log navigation are command line apps that require developers to familiarize themselves with another API.

A simple, easy-to-use GUI tool for displaying and navigating through system log files would benefit developers by allowing them to efficiently find the information they need without requiring advanced querying techniques (such as SQL). In order to be worthwhile, the search functionality needs to be more advanced than mere string matching (which can easily be done through a command line or one of the many existing log-viewing apps.)

The existence of such a tool – simple, with a GUI rich in functionality – is not present as an open-source project.

1.3 Result

EasyLog is a GUI tool for Mac OSX that allows users to search, filter, and highlight operating system logs. It aims to be **intuitive** and **user-friendly**. Although EasyLog was designed on OSX El Capitan, it has several configuration options that make it compatible with any operating system that uses text-based logs.

EasyLog harnesses the power of the Apache Lucene search engine library. It allows users to create sophisticated Boolean queries and conduct simultaneous string searches, substring searches, regex searches, and fuzzy searches with up to 3 keywords. It also allows users to filter files based on the log file's content or the log file's name.

Moreover, EasyLog has a **highlighting** feature. Log files containing the user-specified term are displayed with a different text color. Since the highlighting feature works in conjunction with the filtering feature, the user has numerous options for tailoring the display to fill their needs.

Finally, while EasyLog's default directory is /var/log, this can be changed through the GUI. Users can choose what directory to index, what index to search, whether or not to create an index from scratch, and the maximum number of logs returned with each search query.

1.4 Outline

Section 2 of this report presents background data relevant to EasyLog's implementation, including details about the Lucene library and techniques for parsing log files.

Section 3 describes the final implementation of EasyLog and its software architecture (Sections 3.1 and 3.5). It also provides an overview of EasyLog's main features in Sections 3.2-3.4.

Section 4 examines how the final software differed from its initial conception and evaluates EasyLog in three different ways. First, by measuring the speed of its indexing capabilities and of different search queries. Second, by describing a series of JUnit test cases that were created to ensure that EasyLog functions as described. Third, by summarizing the results of a usability study that determines whether or not EasyLog achieved its goal of being user-friendly.

Section 5 concludes the report. It discusses the relevance of the project and what work could be conducted on EasyLog in the future.

References and Appendices are included at the end of the report.

2. BACKGROUND INFORMATION

Like most operating systems, the format of each log file for OSX El Capital is essentially unique. Frequently, the log is prefixed with a timestamp, although the format of this timestamp also differs across files. For example:

system.log:

```
Dec 10 15:37:13 Patricias-MacBook-Pro kernel[0]: ****
[IOBluetoothHostControllerUSBTransport][ClearFeatureInterruptEndpointHalt] --
successfully posting another read for the mInt0InterruptPipe --
mInterruptPipeInOutstandingIOCount = 1 -- this = 0xf000
```

opendirectoryd.log:

```
2016-11-06 10:31:40.558255 EST - AID: 0x0000000000000000 - opendirectoryd (build
406.10.1) launched...
```

displaypolicyd.log:

```
u>118629309 /usr/libexec/displaypolicyd: Started at Sun Jan 31 21:28:33 2016
```

This makes parsing log files, not to mention automatically extracting interesting data (such as timestamps) challenging.

The parsing strategy used by EasyLog were inspired by **Log File Navigator** [1], an open-source, enhanced log file viewer for OSX and Linux.

Log File Navigator (lnav) creates a custom "configuration file" for each log which contains specific instructions for parsing and extracting information from that particular log file. Many formats are included by default, but users can define their own by creating a JSON file. When loading each log file, lnav iterates through all its different log format options until it finds one that successfully matches the first 1000 lines of the log file.

Log formats are typically defined through sophisticated **regexes** and / or **samples**. For example:

```
  "example_log" : {
    "regex" : {
        "custom1" : {
            "pattern" : "^(?<timestamp>\\d{4}-\\d{2}-\\d{2}T\\d{2}:\\d{2}:\\d{2}\\.\\d{3}Z)
                         <<(?<level>\\w+)--(?<component>\\w+)>>(?<body>.*)$"
             }
          }
```

(Sample code taking from the lnav documentation; see Reference [2])

Information is extracted from log files into SQL tables, which can be queried from the lnav user interface. While relational tables are great for managing and querying large quantities of data, the additional structure makes the both the querying and the parsing of log files more difficult. In addition, the user must be familiar with SQL.

Finally, lnav also has support for decompressing log files and recursively iterating through directories.

Apache Lucene [3] is an open-source, 100%-pure Java library that offers scalable, high-performance indexing. It is free to use (and modify) in commercial software. Lucene is used in Enterprise solutions such as Apache Solr and Elasticsearch due to the library's efficiency and reliability. The key to Lucene's success is its use of an **inverted index**. An inverted index is a data structure that maps content (such as text-based words) to their encompassing document. Each Lucene entry is stored as a *Document*, containing various named *Fields*.

Unlike traditional search engines (such as databases), the inverted index works well with messy, text-based data that doesn't adhere to a strict format. Given that log files meet this criteria, Lucene is a good solution for perusing operating system logs.

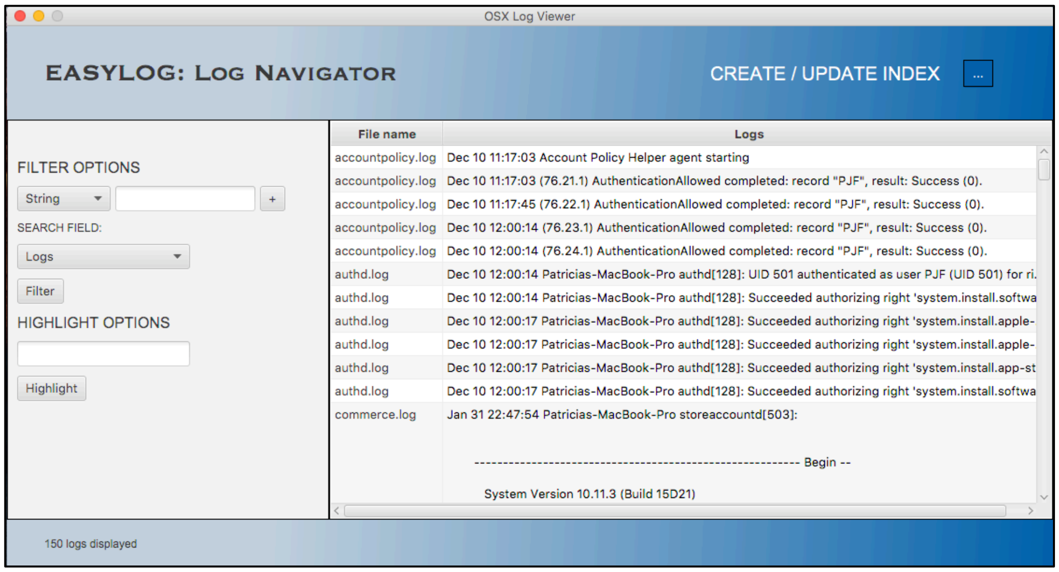
Furthermore, Lucene offers numerous powerful query options: phrase queries, wildcard queries, proximity queries, range queries, fuzzy queries, and more. These can either be constructed from a String using Lucene's **QueryParser**, or constructed from scratch through Lucene's Query API.

Each query is encapsulated in a concrete instance of the abstract Query class, which is extended by a different sub-class for each type of query. Instantiated a Query object simply involves creating a Term object and passing it to the Query constructor. A Term contains the search text and the name of the field that is being searched.

The fancier features of Lucene – faceting, multiple-index searching, ranked searching, result grouping – were not harnessed by EasyLog, but their existence leaves the door open for implementing new sophisticated features.

3. RESULT

The goal of EasyLog is to provide powerful search tools for developers perusing operating system logs, presented through an intuitive interface.



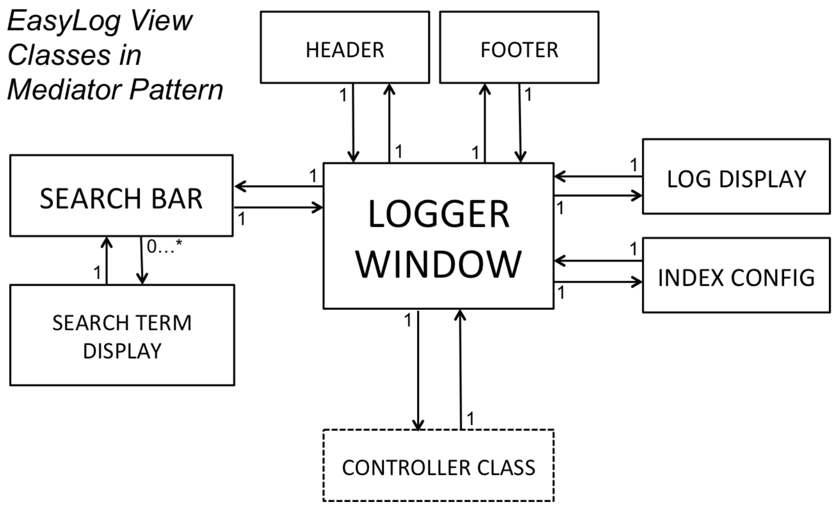
File name	Logs
accountpolicy.log	Dec 10 11:17:03 Account Policy Helper agent starting
accountpolicy.log	Dec 10 11:17:03 (76.21.1) AuthenticationAllowed completed: record "PJF", result: Success (0).
accountpolicy.log	Dec 10 11:17:45 (76.22.1) AuthenticationAllowed completed: record "PJF", result: Success (0).
accountpolicy.log	Dec 10 12:00:14 (76.23.1) AuthenticationAllowed completed: record "PJF", result: Success (0).
accountpolicy.log	Dec 10 12:00:14 (76.24.1) AuthenticationAllowed completed: record "PJF", result: Success (0).
authd.log	Dec 10 12:00:14 Patricias-MacBook-Pro authd[128]: UID 501 authenticated as user PJF (UID 501) for ri.
authd.log	Dec 10 12:00:14 Patricias-MacBook-Pro authd[128]: Succeeded authorizing right 'system.install.softwa
authd.log	Dec 10 12:00:17 Patricias-MacBook-Pro authd[128]: Succeeded authorizing right 'system.install.apple-
authd.log	Dec 10 12:00:17 Patricias-MacBook-Pro authd[128]: Succeeded authorizing right 'system.install.apple-
authd.log	Dec 10 12:00:17 Patricias-MacBook-Pro authd[128]: Succeeded authorizing right 'system.install.app-st
authd.log	Dec 10 12:00:17 Patricias-MacBook-Pro authd[128]: Succeeded authorizing right 'system.install.softwa
commerce.log	Jan 31 22:47:54 Patricias-MacBook-Pro storeaccountd[503]: -- Begin -- System Version 10.11.3 (Build 15D21)

When the software is launched, a Lucene Index is automatically created based on the Operating System's default logging location. For OSX, this is */var/log*. A new index can be created (or updated) at any time by clicking the "Create / Update Index" Button. At the moment, only text-based logs are parsed; ASL files and compressed .gzip files are not extracted.

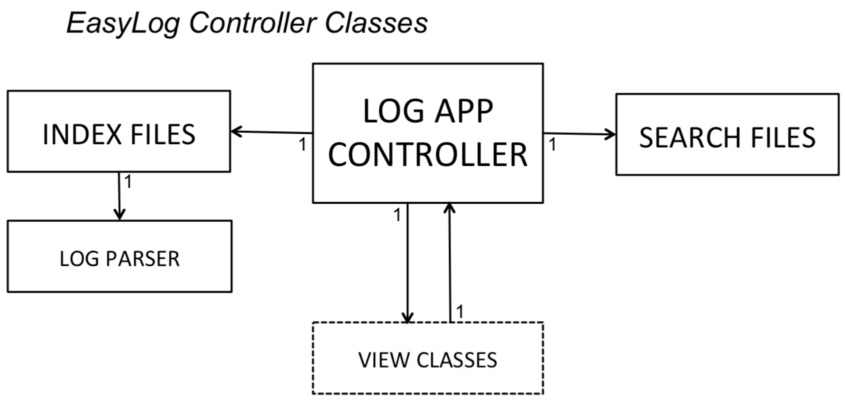
3.1 System Design

EasyLog is a Java application built using a Model-View-Controller (MVC) design pattern. The project is dependant upon 4 jars containing code from the Lucene Library: the Lucene JAR, the queryparser JAR, the common analysis JAR, and the Lucene demo JAR.

The *view* portion is built using the JavaFX platform [4]. As a whole, the view follows a Mediator Pattern, with the LoggerWindow class playing the central role of mediator and communicating with the various other elements of the view, including the search bar, header, and log display.



The *controller* portion is separated into two halves: the IndexFiles and SearchFiles classes, built off of code from the Lucene library. These classes are used for creating and searching through the Lucene index. The *LogAppController* class defines several event handlers that are triggered by user interactions with the GUI, and are responsible for coordinating the index search and returning the results to the view classes.

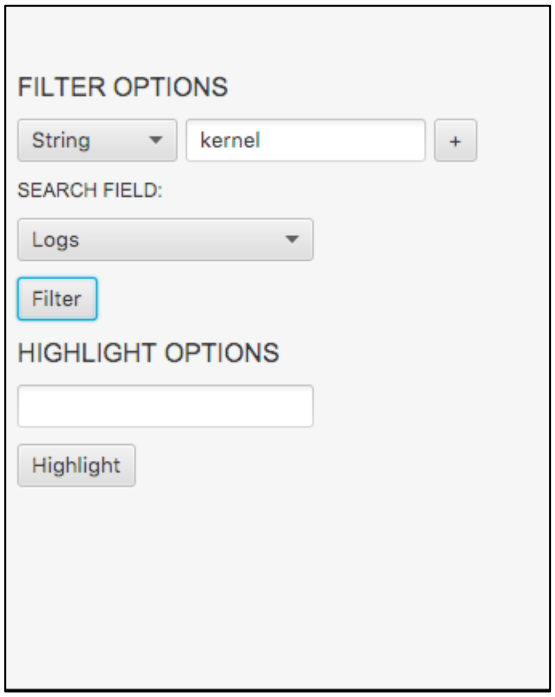


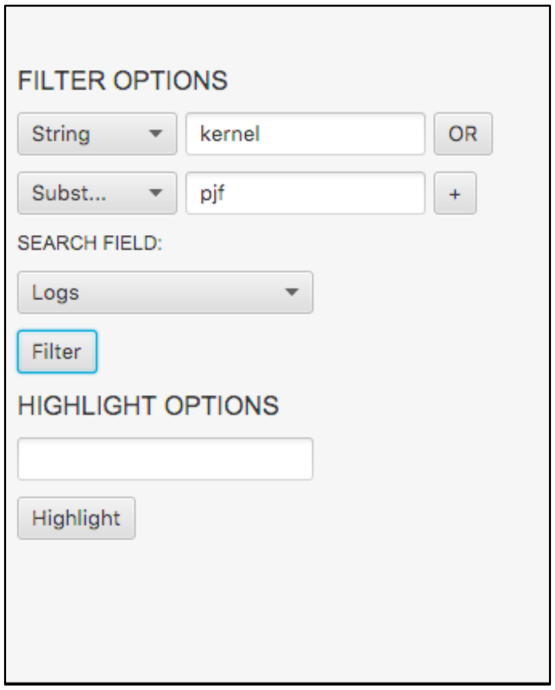
The *model* is comprised of the Lucene index and the log files off which it's based. Each log file is split into multiple Lucene Documents; these Documents contain several searchable fields, the important ones being the "contents" field (used to store the actual log) and the "filename" field (used to store the name of the file from which the log was extracted.)

The index itself (stored at a location defined by the configuration options) is comprised of 4 binary files and a lock file (to coordinate synchronous updates): _10.cfe, _10.cfs, _10.si, segments_11 and write.lock.

3.2 Filtering Feature

Filtering options are displayed in the left-hand menu. Initially, only one text field is displayed. The user can add more fields by clicking the '+' button next to the text field, to obtain a maximum of three filtering terms.





The type of search is controlled through a combo box – the user can conduct a regular string search, a substring search (slower but often more useful), a Regex search, or a Fuzzy search. The button next to the text field controls how the sub-queries are concatenated together in the final Boolean Query. Clicking on an 'AND' button changes it to an 'OR' button, and vice-versa. When the *Filter* button is pressed, a Lucene Query object is created for each term based on its search type: Phrase Queries, Regex Queries, WildCard Queries, and Fuzzy Queries:

```java
PhraseQuery.Builder builder = new PhraseQuery.Builder();
Query q = null;

if (currDisplay.getSearchType().equals("String")) {
   builder.add(new Term("contents", currDisplay.getText()));
   q = builder.build();
 }
```
(Code snippet from the QueryCreator class)

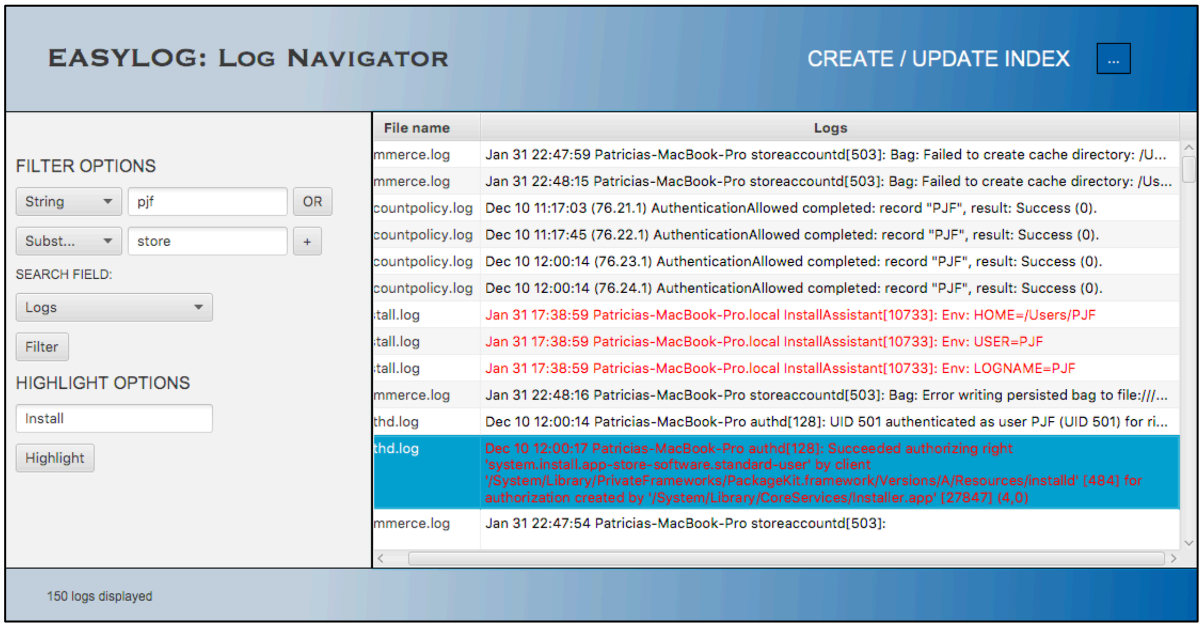
These are bundled together into composite BooleanQueries, which are then passed to the controller object and used to search the Lucene index. Resulting documents are returned to the *LoggerWindow*, which passes them to the *LogDisplay*, which immediately updates its appearance.

If a user wishes to see more of particular log entry, they can click on the table cell to expand it.

3.3 Highlighting Feature

The highlight feature is also accessible through the left-hand menu. There is only one text field, and its searching capacity is case sensitive.

When the *Highlight* button is clicked, the displayed logs are perused one by one. Any entry containing the specified substring is colored a bright red to make it easy to see. The highlight feature was implemented by overriding the *updateItem* method of the columns' TableCells.



String: pjf
Subst...: store
SEARCH FIELD: Logs
Install

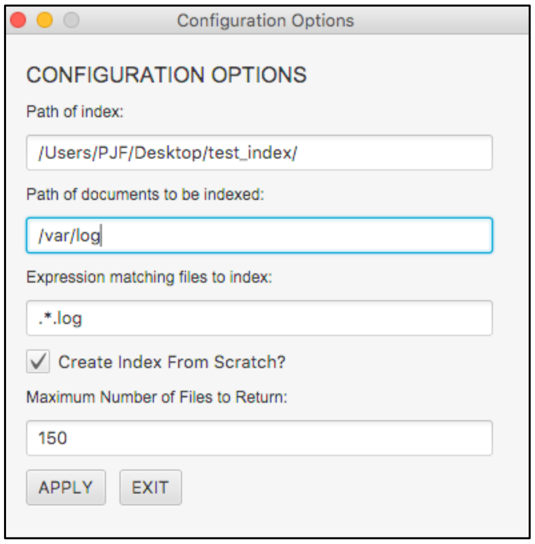
File name	Logs
mmerce.log	Jan 31 22:47:59 Patricias-MacBook-Pro storeaccountd[503]: Bag: Failed to create cache directory: /U...
mmerce.log	Jan 31 22:48:15 Patricias-MacBook-Pro storeaccountd[503]: Bag: Failed to create cache directory: /Us...
countpolicy.log	Dec 10 11:17:03 (76.21.1) AuthenticationAllowed completed: record "PJF", result: Success (0).
countpolicy.log	Dec 10 11:17:45 (76.22.1) AuthenticationAllowed completed: record "PJF", result: Success (0).
countpolicy.log	Dec 10 12:00:14 (76.23.1) AuthenticationAllowed completed: record "PJF", result: Success (0).
countpolicy.log	Dec 10 12:00:14 (76.24.1) AuthenticationAllowed completed: record "PJF", result: Success (0).
tall.log	Jan 31 17:38:59 Patricias-MacBook-Pro.local InstallAssistant[10733]: Env: HOME=/Users/PJF
tall.log	Jan 31 17:38:59 Patricias-MacBook-Pro.local InstallAssistant[10733]: Env: USER=PJF
tall.log	Jan 31 17:38:59 Patricias-MacBook-Pro.local InstallAssistant[10733]: Env: LOGNAME=PJF
mmerce.log	Jan 31 22:48:16 Patricias-MacBook-Pro storeaccountd[503]: Bag: Error writing persisted bag to file:///...
thd.log	Dec 10 12:00:14 Patricias-MacBook-Pro authd[128]: UID 501 authenticated as user PJF (UID 501) for ri...
thd.log	Dec 10 12:00:17 Patricias-MacBook-Pro authd[128]: Succeeded authorizing right 'system.install.app-store-software.standard-user' by client '/System/Library/PrivateFrameworks/PackageKit.framework/Versions/A/Resources/installd' [484] for authorization created by '/System/Library/CoreServices/Installer.app' [27847] (4,0)
mmerce.log	Jan 31 22:47:54 Patricias-MacBook-Pro storeaccountd[503]:

150 logs displayed

The highlight feature can be used in conjunction with the filtering feature.

3.4 Indexing Options

The indexing options are accessed through the *Extra Options* button in the top right-hand corner. Clicking it brings up the following interface:



CONFIGURATION OPTIONS

Path of index: /Users/PJF/Desktop/test_index/

Path of documents to be indexed: /var/log

Expression matching files to index: .*.log

✓ Create Index From Scratch?

Maximum Number of Files to Return: 150

A combination of text fields and check boxes allow a user to choose what directory they would like to index, and (if they choose to create multiple Lucene Indexes) which one they would like to search. By default, Lucene scraps its old index and builds an entirely new one instead of updating; this can also be changed. Finally, a user can choose which log files they would like parsed and added to the index. The default value is **.log*, encompassing 13 of the different log files on OSX.

3.5 Log Parsing and Document Creation

Since the format of each log file is different, the formats must be known in advance for effective parsing. Otherwise, a generic parsing strategy is applied, with potentially substandard results.

A separate LogParser class was created to deal with this amount of variation. The lines of each file are read in individually, separated by newline characters. Unfortunately, a newline character does not necessarily indicate the beginning of a new log entry.The LogParser.*isNewLog* method was created to detect if the line contains the start of a new log entry, thus encapsulating the messy process of dealing with the variation of individual log files.

Each line is appended to a String (currLog) until isNewLog returns true, at which point a new Document is created and the entire process starts again:

```
while (true) {
    String line = txtReader.readLine();
    if (line != null) {
        if (isNewLog(line, filename)) {
         // create a document with the previously aggregated log entry
             if (currLog != null) {
                 Document newLog = new Document();
                 newLog.add(pathField);
                 newLog.add(new StringField("filename", filename, Field.Store.YES));
                 newLog.add(new TextField("contents", currLog, Field.Store.YES));
                 IndexFiles.addDocToIndex(writer, newLog, file);
                 counter++;
              }

          // start aggregating a new log entry
           currLog = line;
      } else {
         // since the segment read is just a piece of a log, add it to current log
           currLog += ("\n" + line);
    }
}
```
(Code snippet from the LogParser class, lines 33-50)

Luckily, since the EasyLog prototype only supports a limited amount of log files (most of which have a new log entry at every newline character) this problem proved to be trivial; it was sufficient to check the first few characters of each line to deduce if it was a timestamp, and thus a new log entry.

While this is a decent solution for a working prototype, it is not robust and certainly not extensible. A better system would involve automatically detecting the file type and then using regexes to determine where logs start and end, much like the Log File Navigator software.

4. EVALUATION

4.1 Performance Evaluation

Since log files can be massive, it is important that EasyLog is capable of building indexes and conducting searches in a timely manner.

A timing mechanism came built-in with the Apache Lucene classes *IndexFiles* and *SearchFiles*, using calls to the Java method Date.getTime()

```
Date end = new Date();
System.out.println(end.getTime() - start.getTime() + " total milliseconds");
```
(Snippet from the IndexFiles class)

The program essentially creates Date objects at the beginning and end of the indexing (or searching) operating and then prints the time difference to the console.

Table 1.1. Timing of Index Operations

NUMBER OF FILES	TOTAL SIZE (MB)	DOCUMENTS CREATED	AVERAGE TIME (MS)	STANDARD DEVIATION (MS)	MIN TIME (MS)	MAX TIME (MS)
13	96	955	530	40	481	619
39	289	2865	660	82	601	860

96 MB is the size of all text-based log files taken from a sample system. 290 MB is the size of all logs in the /var/log directory (including ASL files and compressed folders). This indicates that EasyLog has a reasonable indexing speed in typical cases, and seems to perform decently when scaled. Note that none of these cases test EasyLog in extreme cases (multiple GBs or hundreds of files). EasyLog does not claim to be an Enterprise solution, and these test cases cover the functionality that typical users will need.

Table 1.2 Timing of Search Operations

TYPE OF SEARCH	KEYWORD	NUMBER OF DOCUMENTS	DOCUMENTS RETURNED	AVERAGE TIME (MS)
String Search	store	961	25	1.3
	kernel	961	92	0.5
Regex Search	[st]tore	961	25	0.5
Substring Search	store	961	142	5.3
	kernel	961	95	0.2
Fuzzy Search	auths	961	9	15.5

The average time is based on a rapid succession of 1000 queries. All of these entries are highly satisfactory for a number of documents involved and the size of log entries. Even substring searches and fuzzy searches – unquestionably the most time consuming – will not force a user to wait long periods of time for results. In fact, updating the UI takes far longer and makes the search time, by comparison, seem negligible.

4.2 Use Cases

A series of J-Unit test cases were written in order to ensure that all features of EasyLog work as presented. A small sample data set was created based on real log values; this ensured that the results of all test cases were known in advance. The data set consists of 3 log files: **wifi_test.log**, **system_test.log**, and **commerce_test.log**. They contain a total of **15** documents.

Unit tests include:

- **IndexingTest**: tests indexing all logs documents, and indexing a subset of logs (when the filenames are restricted).

- **BasicSearchTest**: tests each type of search (string, substring, regex, and fuzzy) with a single keyword. Also testes searching on filename vs. log content

- **SearchTests**: tests all Boolean combinations of searches with 2 terms (AND, OR), and 3 terms (AND/AND, AND/OR, OR/AND, OR/OR). Unit tests were also added for cases where the text fields were empty.

All unit tests pass, indicating that EasyLog works as presented.

4.3 Usability Study

One of the primary goals of EasyLog is to be user friendly and more accessible than a command line interface. To test whether or not EasyLog achieved this goal, a small usability study was conducted on 6 participants of various technical and non-technical backgrounds. After having a chance to user the software and ask questions, they then filled out a page-long questionnaire.

In the first part, they gave a rating from 1-5 (1 being very bad, 5 being very good) to the following characteristics of EasyLog: **Ease of Use, Visual Appeal, Efficiency, and Usefulness**. The mean and standard deviation of each value is as follows:

Attribute	Mean Score	Standard Deviation
Ease of use / user friendliness	4.2	0.14
Visual Appeal	4.7	0.22
Efficiency	4.7	0.22
Usefulness	4.5	0.58

The second part of the questionnaire asked general, open-ended questions (see *Appendix 1*):

- What is your overall impression of the software?
- Are there any elements of the UI that you found confusing, or that worked in a way you didn't expect?
- Do you (as a computer user or a developer) think the software could be useful? Why or why not?
- What additional features would you like to see in similar software?
- Do you have any suggestions about how the software could be improved?

Overall, users are pleased with the design of the GUI and the functionality of EasyLog. They agree that EasyLog has potential to be a useful tool, although the UI could use a few minor tweaks and additions to really make it intuitive and user-friendly. A score of 4.2 in "Ease of Use" and 4.7 in "Visual Appeal" indicates that EasyLog generally fulfills its objectives.

Broadly speaking, users think that EasyLog was a huge improvement to the command line interface, especially since it consolidates multiple files into a single viewer. A couple testers think it could encourage developers to use log files more, while others dismiss EasyLog as being useful only to a niche market.

Most people are satisfied with the range of query options, although almost all (4/6) are confused by the difference between 'String search' (searches for a whole word) and a 'Substring search' (searches for part of a word). This unintuitive search behavior is a consequence of Lucene's implementation of string search, which is intended for full-string searches and optimized with this in mind. Substring searching is considerably slower, especially as indexes grow. However, users point out that substring searching is the default standard in nearly every other application.

Testers suggest making 'Substring search' the default, or removing the 'String search' entirely. While people agree that 'Regex search' and 'Fuzzy search' could be useful in specialized cases, the vast majority of users are only interested in substring searches. No one expressed a desire for the ability to use more search terms, or were off put by the mechanism to concatenate terms into Boolean queries.

Two additional features that were requested several times was the ability to sort by date/time, and the ability to open up a log file by double clicking on an entry. Users also note that the ability to expand a log file by clicking on it was not intuitive. Finally, 2 different users recommend that case-sensitivity be removed in all cases (both for filtering and highlighting), while 1 user thinks that case-sensitivity was useful for the highlight feature.

During the user evaluation process a bug involving blank search terms was discovered and has been addressed. The Regex query also didn't work as users expected when wildcards (*, ?) were involved; this is likely another consequence of Lucene's implementation, which seperates Regex and Wildcard queries into distinct entities. In order to make the user experience smoother, a mechanism for differentiating these queries should be added to the LogParser class.

Other edge cases – such as searching for special characters or sentences – are not covered by EasyLog and did not come up in the user evaluation. Although they must be considered in a final product (for robustness and quality), such features did not seem to be desired by users and their exclusion from the prototype was not even noted.

5. CONCLUSION

5.1 Summary

In sum, EasyLog helps developers troubleshoot computer problems by providing an easy-to-use window into system logs. By offering filtering, highlighting, and index configuration features, users are able to customize the GUI into displaying exactly what log information they need to see.

5.2 Relevance

Logging is crucial for proper operating system diagnostics. Not only are log formats OS-specific, but their interpretation and manipulation requires a modicum of understanding of operating system functionality. In short, EasyLog is a useful tool for people working closely with OSX and having to wade through massive amounts of log information.

5.3 Future Work

The most important improvements to EasyLog involve implementing the suggestions from the usability study: defaulting to substring search, enabling sorting by date, adding a double click functionality, and adding tooltips or instructions to EasyLog's various features. This would ensure that EasyLog fulfills its initial criteria of being simple and user-friendly.

In addition, the initial creation of the Lucene index is very slow. Since it is necessary to open up every log file and read in the text content one line at a time, there isn't a way to magically make this efficient. However, it would be nice for users to be aware that this process is slow by nature, and to receive a visual cue when the index is being created (for example, a loading bar). Ideally, it would be beneficial to have EasyLog running in the background and constantly updating the Lucene index. Then, when the user fires up the GUI, there is no wait time.

One potential workaround is to have the operating system write logs to the Lucene index directly. For example, the software **Loggly** [5] uses the open-source log collector *fluentd* to act as a go-between for the OS and their server. There are also system configuration files for logs (located in the /etc directory on OSX) that could be modified to obtain the desired functionality. This type of arrangement would eliminate the need for many of the messy parsing methods and help EasyLog update in real time.

Another important step is to make EasyLog compatible with multiple file types, specifically binary formats like ASL files, and to enable it to unzip compressed files and access the content within. This will provide a much greater range of information to users.

Furthermore, using more complex regular expressions to extract information from log files will help create a richer index. This in turn opens up more search options and configuration options for EasyLog users – particularly the requested date / time search.

Finally, EasyLog could be modified to provide an unlimited number (or a very large finite number) of fields for both the filtering and highlighting features. It could also be equipped with advanced features such as bookmarking and remote viewing.

Contributions of Team Members

Patricia Foster (100883992) was the sole developer of the project: research, conception & design, implementation, testing, and report writing.

REFERENCES

[1] Log File Navigator, “The Log File Navigator.” http://lnav.org/ (Accessed December 14th)

[2] Log File Navigator, “Log Formats” http://lnav.readthedocs.io/en/latest/formats.html#log-formats (Accessed December 14th)

[3] The Apache Software Foundation, “Apache Lucene Core.” http://lucene.apache.org/core/ (Accessed December 14th)

[4] Oracle, “JavaFX: Getting Started with JavaFx.” https://docs.oracle.com/javase/8/javafx/get-started-tutorial/jfx-overview.htm (Accessed December 14th)

[5] Loggly Inc., “Mac OS X Logs.” https://www.loggly.com/docs/send-mac-logs-to-loggly/ (Accessed December 14th)

APPENDIX 1: User Evaluation Questionnaire

On a scale of 1-5 (1: very bad, 3: average, 5: excellent), how would you rate the following attributes?

ATTRIBUTE	SCORE
Ease of use / user friendliness	
Visual Appeal	
Efficiency	
Usefulness	

What is your overall impression of the software?

Are there any elements of the UI that you found confusing, or that worked in a way you didn't expect?

Do you (as a computer user or a developer) think the software could be useful? Why or why not?

What additional features would you like to see in similar software?

Do you have any suggestions about how the software could be improved?

Thank you for your time!