

# **Text Transformation Project Report**

CS 4624 Virginia Tech, Blacksburg, VA

2/27/2015

Professor: Edward A. Fox

Client: Nathan Hall

Developers: Stephen Fenton, Dustin Thompson, Zach Henke, Kevin Cox

# Table of Contents

Introduction	Page(s)
1. <i>Executive Summary</i> .....	4
User's Manual .....	5-6
1. <i>Regular Expression Tree Usage</i> .....	5
2. <i>Text Transformation GUI Usage</i> .....	5
3. <i>Alternate Parsing Tool Usage</i> .....	6
Developer's Manual .....	7-31
1. <i>Requirements</i> .....	7-10
1.1) <i>Project Deliverables</i> .....	7
1.2) <i>Development Methodology</i> .....	8
1.3) <i>Development Tools</i> .....	9
1.4) <i>Preliminary Timeline</i> .....	9
1.5) <i>Project Extensions</i> .....	10
1.6) <i>VTechWorks Deliverables</i> .....	10
2. <i>Backend Design</i> .....	10-12
2.1) <i>Introduction</i> .....	10
2.2) <i>Tools used</i> .....	10
2.3) <i>OO Design</i> .....	10
2.4) <i>Parsing</i> .....	11
2.5) <i>Error Catching</i> .....	12
2.6) <i>Alternatives</i> .....	12
3. <i>Frontend Design</i> .....	12-15
3.1) <i>User Interface</i> .....	12
3.2) <i>Java Code Design</i> .....	13
3.3) <i>UI Flow Model</i> .....	13
4. <i>Implementation Plan</i> .....	16-18
4.1) <i>Programming Language</i> .....	16
4.2) <i>Tools Employed</i> .....	16
4.3) <i>Parsing Strategy</i> .....	16
4.4) <i>Source Control</i> .....	17
4.5) <i>Agile Progress Tracking</i> .....	17
4.6) <i>Code Review</i> .....	18
4.7) <i>Timeline</i> .....	18
4.8) <i>Roles</i> .....	19
5. <i>Prototyping</i> .....	20-29
5.1) <i>Regular Expression Tree</i> .....	20

5.2) <i>Tree Tool</i> .....	20
5.3) <i>Binary Data Structure</i> .....	22
5.4) <i>Citation Parser Tool</i> .....	24
5.5) <i>Refinement 1</i> .....	26
5.5) <i>Refinement 2</i> .....	27
6. <i>Testing</i> .....	29
6.1) <i>Software-based Unit Testing</i> .....	29
6.2) <i>Manual Interface Testing</i> .....	30
Lessons Learned.....	31
1. <i>Problems Encountered</i> .....	31
2. <i>Future Work</i> .....	31
Acknowledgements.....	32
References .....	32

# **Introduction:**

## **1) Executive Summary**

The purpose of this project is to assist the VTTI in converting a large citation file into a CSV file for ease of access. It required us to develop an application which can parse through a text file of citations, and determine how to properly put the data into CSV format. We designed the program in Java and developed a user-interface using JavaFX, which is included in the latest edition of Java.

We came up with two main tools: the developer tool and the parsing program itself. The developer tool is used to build a tree made up of regular expressions which would be used in parsing the citations. The top nodes of the tree would be very general regexes, and the leaf nodes of the tree would become much more specific. This program can export the regex tree as a binary file which will be used by the main parsing program.

The main parsing program takes three inputs: a binary regex tree file, a citation text file, and an output location. Once run, it parses the citations based off of the tree it was given. It outputs the parsed citations into a CSV file with the citations separated by field. For any citations that the program is unable to process, it dumps them into a failed output text file so.

We also created an additional program as an alternative solution to ours. It uses Brown University's FreeCite parsing program, and then outputs parsed citations to a CSV file.

## **User Manual:**

### **1) Regular Expression Tree Usage**

The tree tool is a developer tool used to build a regex tree which the main program uses to parse citations. Upon starting this program, you will see an empty tree, as well as having the option to import or export a tree.

When dealing with an empty tree, you will see a bar labelled New Regex Node. You may click it to fill in what the new node will be. The top bar is the regular expression, and the bottom bar is for the field and capturing group. As an example, you could put in `((?:a-zA-Z))+,s(?:0-9)+` as your regex, and Author,Year as the capturing groups. This regex would be capable of parsing something such as "Bob, 2005" and it would designate Bob as the author, and 2005 as the year. Once you have entered everything you want for your regex node, press enter and it will be entered into the tree. Now you can export this tree file to be used by a citation parsing program. To do this press file and export, and you will be able to name the file which will be saved.

You can also add child nodes to a node. Right click a node and press add child node. It will create a new regex node which you can enter in another regex into. When creating a tree, have the higher level nodes be more general, and have the lower level nodes be more and more specific.

You can also remove leaf nodes. To do this, right click on a leaf node and press remove leaf node. It will be deleted.

To import a tree, press file and import and it will open up your file explorer. Select a .RGXT file which was created using this tool.

### **2) Text Transformation GUI Usage**

The Text Transformation program's interface takes 3 main inputs. The first is used to select which binary regex tree the parsing program will use. The second input is to select which file you want to parse. You must ensure that it is a plain text file. The third thing you must select is the output destination. Once you run the file, the output csv file as well as the failed output text file will be placed in this destination.

After you have selected all 3 files/directories, you will be able to press the start button to begin running the parsing program. You will see a progress bar which shows you how far along the parsing is. It will increment forward for every citation it hits or misses.

Once it has completed parsing the citations, it will produce two output files. The first is a csv file which contains citations which were successfully hit by the parsing program. Each citation will be broken up into different fields. The second output file is a text file which contains the citations the parser wasn't able to match. If you end up with a large failed output file, the regex tree may need to be refined to parse the file better.

The output csv file will be labeled as "parsed\_citations[timestamp].csv"

The failed output file will be labeled as "failed\_citations[timestamp].csv"

The timestamp will be in the format yyyy-MM-dd--hh-mm-ss

### **3) Alternate Parsing Tool Usage**

This program simply takes 2 inputs. The first is the citation file you want to parse. It needs to be a text file. The second is the output destination. Once you have entered both files/directories, press run and you will see the number of citations it has parsed out of the number there are total. After it has finished running you will get an output csv file with the citations separated by field.

# **Developer's Manual:**

## **1) Requirements**

Outlined here are the methodologies we will use to develop our text transformation tool, our overall goals for the project and end output, and ways we intend to at least explore extending the project to facilitate other institutes within the University. We will define individual roles and goals for these jobs, set a preliminary timeline for deliverables, and detail any other mechanisms for ensuring a successful product.

### **1.1) Project Deliverables**

We are working with the Virginia Tech Transportation Institute<sup>2</sup>, or VTTI<sup>2</sup>, to help organize their citation data. The primary goal of this project is to take in a text file consisting of a variable number  $x$  of citations and transform them into a comma-separated value (henceforth referred to as .CSV) file. Each citation is separated from other citations by a newline character. Examples of citation inputs and expected outputs can be seen in **Figure 1a** and **Figure 1b**. These citations are not guaranteed to fall into one category, and even citations of the same format can have mild variations that could cause parsing to fail. Therefore, it will be in our best interests to develop one of the following tools:

- A. A robust, extensible regular expression suite that will allow us to read through the file and get exact matches to specific regular expression types.
- B. A tool that utilizes pre-existing natural language tools to help in our analysis of the text.

Both of these tools will present their own advantages and disadvantages in terms of development effort, usability, etc; as an example, the regular expression example is quicker to develop, but the processing time could potentially become very large, and without an extensible design, adding regular expressions could be a tedious task.

We also intend on delivering a user interface that will allow users to select a text input file and output directory for a created file. This interface will also be responsible for displaying any errors that our backend parser runs into. Additional options may be added as development furthers and requirements are tweaked or altered.

Finally, we will be delivering a robust test suite that will cover a variety of cases that may affect performance and/or correctness of the utility. All code will be covered by these tests; the user interface will be manually tested to ensure correctness. These tests will run the gamut of anything from a null input to an attempted memory overflow in order to ensure the best possible product for our client.

Citation Examples
Aatique, M., Mizusawa, G., and Woerner, B. (1997). Performance of hyperbolic position location techniques for code division multiple access, Proceedings of Wireless '97, Calgary, Canada, July 9-11, 1997.
Ahmadian, M. and Ahn, Y.K. (2000). "Performance Analysis of Magneto-Rheological Mounts," Journal of Intelligent Material Systems and Structures, Vol. 10, No. 3, March, pp. 248-256.

**Figure 1a.** Shown above are examples of citation inputs.

CSV Output Examples
"Aatique, M.  Mizusawa, G.  Woerner, B. ",1997,Performance of hyperbolic position location techniques for code division multiple access,Proceedings of Wireless '97,"Calgary, Canada","July 9-11, 1997",,,
"Ahmadian, M.  Ahn, Y.K. ",2000,Performance Analysis of Magneto-Rheological Mounts,Journal of Intelligent Material Systems and Structures,,,10,3,248-256

**Figure 1b.** Shown above are the CSV outputs expected for the citation examples in Figure 1a. Double quotes are used to bypass reserved characters. The CSV columns are ordered as follows: Author, Date, Title, Publication, Location, Date, Volume, Issue, Pages

## 1.2) Development Methodology

For the Text Transformation tool, we will be using an agile methodology<sup>8</sup>. Agile is defined as such:

**Agile development** is an alternative to traditional project management where emphasis is placed on empowering people to collaborate and make team decisions in addition to continuous planning, continuous testing and continuous integration.

What this entails to the client is that we will not blindly come back with a product that will not match their needs; we will return iteratively with both minor and major improvements so that we can work out any issues together and have the product that they desire completed at the end of the project timeline.

We intend to meet up with our client after an initial month of work, barring any immediate need for a meeting, followed up by bi-weekly meetings where we showcase improvements and request feedback or ideas to implement new features.

While agile development methods traditionally employ a Scrum master and have daily standups, due to the nature of the project and the distance of the team members, we will not hold traditional standups and instead communicate



during in-class sessions, through e-mail, and using messenger applications if the need arises.

### 1.3) Development Tools

For this project, we have decided to develop our tool in Java. Java is a portable language that will allow our utility to be run across a variety of devices with Windows, Mac, and Linux being the primary targets. It is a well-known and globally-accepted language used often by each of our team members. Java also allows for an easy User Interface (GUI) design using the JavaFX<sup>3</sup> and Swing libraries.

We will develop our project in the Eclipse<sup>4</sup> development environment (IDE), which all team members have used for 2+ years. Eclipse<sup>4</sup> allows for easy unit testing, helpful debugging, and easy integration with any other Java tools if we need to use them. Code will be stored both locally on team member machines, and in a public GitHub<sup>5</sup> repository where we can all collaborate on code.

### 1.4) Preliminary Timeline

<b>Weeks 1-4</b>	<ul style="list-style-type: none"> <li>• Finish the wiki write up and get it signed by our client and Dr. Fox. (Completed)</li> <li>• Setup code repositories and tools required for development. (Completed)</li> <li>• Code first iterations of text parsing utilities. (Completed)</li> <li>• Set up an initial user interface.</li> <li>• Set up testing classes and initial tests for the utility. (Completed)</li> <li>• Deliver an initial working demonstration to the client by March 20th.</li> </ul>
<b>Weeks 5-6</b>	<ul style="list-style-type: none"> <li>• Add in more parsing utility code to improve citation coverage.</li> <li>• Add error output code to parser.</li> <li>• Add error output to user interface.</li> <li>• Increase code coverage of tests.</li> </ul>
<b>Weeks 7-8</b>	<ul style="list-style-type: none"> <li>• Further parser improvement.</li> <li>• Further test coverage to go with improvement.</li> <li>• Focus on improving performance if necessary.</li> </ul>
<b>Weeks 9-10</b>	<ul style="list-style-type: none"> <li>• Finalize project for client.</li> <li>• Final push for improved performance.</li> <li>• Assure that the project will run on different platforms as expected.</li> </ul>

**Figure 2.** Shown above is the timetable to be followed for the project's duration.

This timeline, as one should expect, is subject to change and new goals may be added (e.g., we have discussed the possibility of a website with the client).

### 1.5) Project Extensions

While we have laid out a solid plan for our utility, we certainly can extend upon it, as we have discussed with our client. We anticipate having a solid, functional, and extensible utility ready for our client at the end of the timeline, but in the event that we have more time or resources than we anticipate, we have several ways to extend the project:

- **Working with other institutes.** While this project focuses solely on the output of files submitted by VTTI<sup>2</sup>, we have been told that working with VBI would be a natural next step to improving our project.
- **Creating a website.** Having the program posted on a website to use would increase ease-of-use for any users unfamiliar with the technical requirements (having Java installed) of running our utility.

These ideas and more can be adapted into our project, and this space may expand as we come up with other ways to improve our project and leave it long-lasting after we have departed the University.

### 1.6) VTechWorks Deliverables

The following is a list of files delivered to VTechWorks for consumption:

***CitationParser.jar***: An executable Java JAR file that transforms text citations.

***TreeTool.jar***: An executable Java JAR file for regex tree developers.

***CitationParserProject.zip***: A zip file containing all source code for the project.

***FinalReport.pdf/docx***: A PDF file and a DOCX file comprised of this document.

***FinalPresentation.pdf/pptx***: A PDF file and a PPTX file of our final presentation.

***VTTI\_clean.txt***: A text file comprised of citations to parse.

***regex\_tree.rgxt***: A sample tree to parse VTTI\_clean.txt.

***FreeCiteTool.zip***: A side tool for non-specific parsing using Brown's FreeCite API.

## 2) Backend Design

### 2.1) Introduction

The problem we need to solve is parsing through a large list of citations, several of them having variations, making them different from each other. To

tackle this, we are going to program in Java and use an object-oriented approach to help separate the different fields in each citation.

## 2.2) Tools Used

Our IDE of choice is Eclipse<sup>4</sup>. Eclipse<sup>4</sup> has several nice features for programming in Java, as well as integration with GIT, so we will easily be able to keep our program in source control. This is vital since we are working in a medium sized group of 4 people.

## 2.3) OO Design

We will parse the document of citations and store them as citation objects, each one containing different fields for all of the members of a citation. For example, the citation class would hold several different fields such as author, title, etc.

```
public class Citation {
    public String Authors, Date1, Title, Publication, Location, Date2, Volume,
        Issue, Pages;

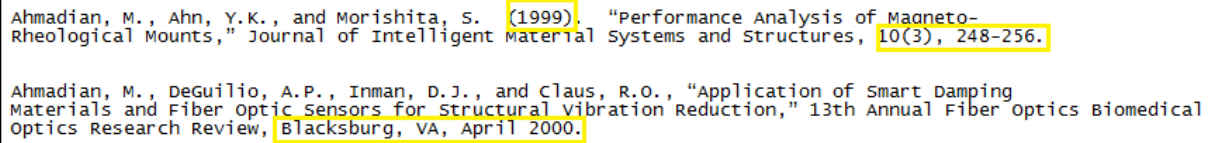
    public Citation() {
        Authors = "";
        Date1 = "";
        Title = "";
        Publication = "";
        Location = "";
        Date2 = "";
        Volume = "";
        Issue = "";
        Pages = "";
    }
}
```

**Figure 3.** Shown above is an example Java Class for storing citations after parsing.

With this framework in place, we would then write the Citations object, in CSV format, to the file specified by the user.

## 2.4) Parsing

The parsing will be the most difficult part of this project. The reason for this is that all of the citations don't follow an exact standard, so there are several variations between them. To manage this, we will have to come up with a complex set of regular expressions which can identify structure, keywords, phrases, and certain special characters to determine which fields are which. Some of the citations even leave out critical information such as the authors and dates.



Ahmadian, M., Ahn, Y.K., and Morishita, S. (1999). "Performance Analysis of Magneto-Rheological Mounts," Journal of Intelligent Material Systems and Structures, 10(3), 248-256.

Ahmadian, M., DeGuilio, A.P., Inman, D.J., and Claus, R.O., "Application of Smart Damping Materials and Fiber Optic Sensors for Structural Vibration Reduction," 13th Annual Fiber Optics Biomedical Optics Research Review, Blacksburg, VA, April 2000.

**Figure 4.** Shown above are example citations highlighting variations in formatting.

Here, you can see that these two citations, while similar, have variations. The first contains a date in parentheses near the beginning, as well as page numbers, while the second only has a publication date at the end of the citation, without any page numbers. These variations make parsing very tricky because of how many there are. Since there are so many citations to deal with (approximately 1500), we will work on parsing smaller chunks of citations at a time. The more we add into the parsing logic, the fewer unparsed citations there will be, until we manage to cover 100% of the citations. While working through it, we will have a way to display the ones not yet parsed, so that we know what we have to focus on getting next.

## 2.5) Error Catching

An important design choice to make is how to handle the parser running into errors. One initial thought is to have a separate file which will print out all of the citations it couldn't successfully parse. This way we will be able to see what our code needs to handle next. Also, It allows the user to see if the remaining few would be something that they just need to do by hand. We would know if a specific citation could not be parsed correctly if it does not match any regular expression used in the parser. A potential problem could be that a citation could match a regular expression but not be correct. That is why we plan on being very specific with our regular expressions so we do not run into that problem.

Every single parse will return a key-value pair of a boolean and a string. The boolean will indicate if the parse was successful (i.e., a leaf-level matching regular expression in the tree was found), and the string will represent either the rearranged and corrected CSV-format string, a critical (unhelpful) error message in event of emergency, or a representation of the closest regular expression match that was found.

## 2.6) Alternatives

An alternate solution to this problem would be to find an open source program with some similar parsing logic already built in, and then adjusting it to fit our needs. Unfortunately, finding something like this has proven difficult, and the ones we have seen so far are in languages not compatible with Java For example, FreeCite<sup>7</sup>, an open-source citation parser from <http://freecite.library.brown.edu/> does something similar to what we need, but it is in Ruby which isn't compatible with our strategy.

### 3) Frontend Design

#### 3.1) User Interface

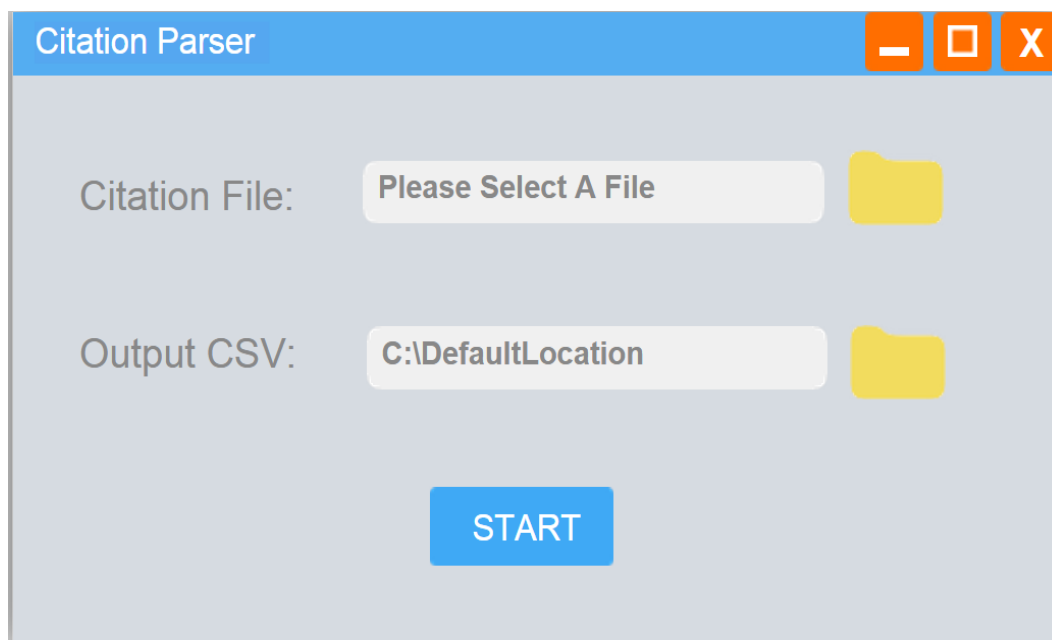
The planned user-interface would be developed in JavaFX<sup>3</sup>, a GUI framework provided with Java SDK 8. JavaFX<sup>3</sup> allows for quick and simplified UI development, by providing a multitude of graphics object libraries as well as allowing the integration of CSS files. The GUI will be responsible for letting the user select a Citation Text File to parse and a location to output a CSV file with the parsed citation data. It is also important that the UI alert the user to critical program errors as well as file reading/writing errors. In addition, the UI will alert the user if the parser had to create a log file containing any citations that could not be parsed. If a Log file is produced it will be provided in the same folder location as the CSV output file.

#### 3.2) Java Code Design

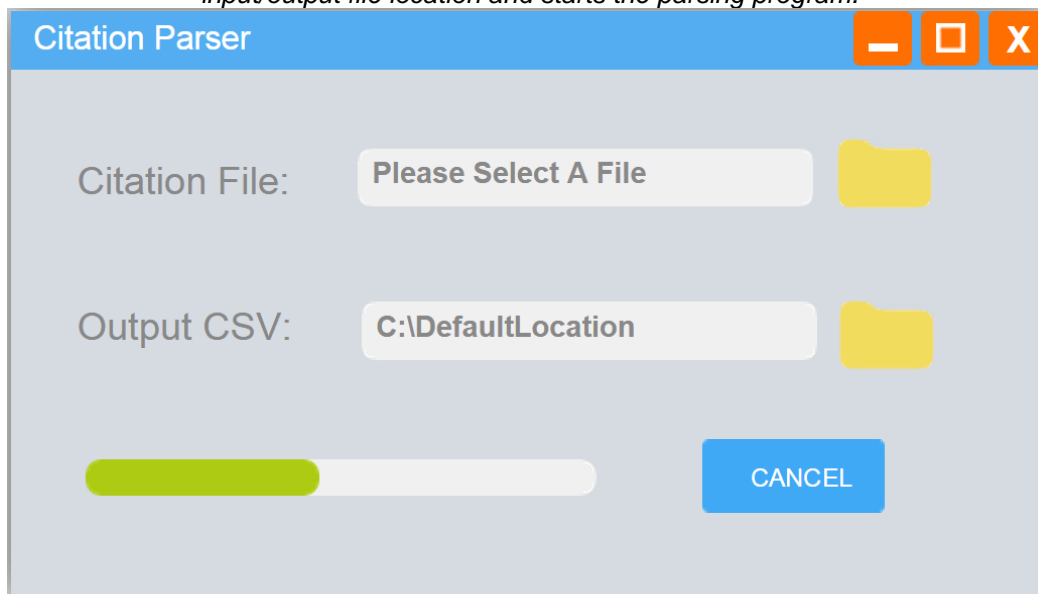
The front end will branch off from the main program thread on program start up to run on a separate application thread. The application thread will ensure that the GUI does not pause or hang while the main thread deals with parsing computations. The front end will communicate with the backend models through standard action Listeners and Observers.

#### 3.3) UI Flow Model

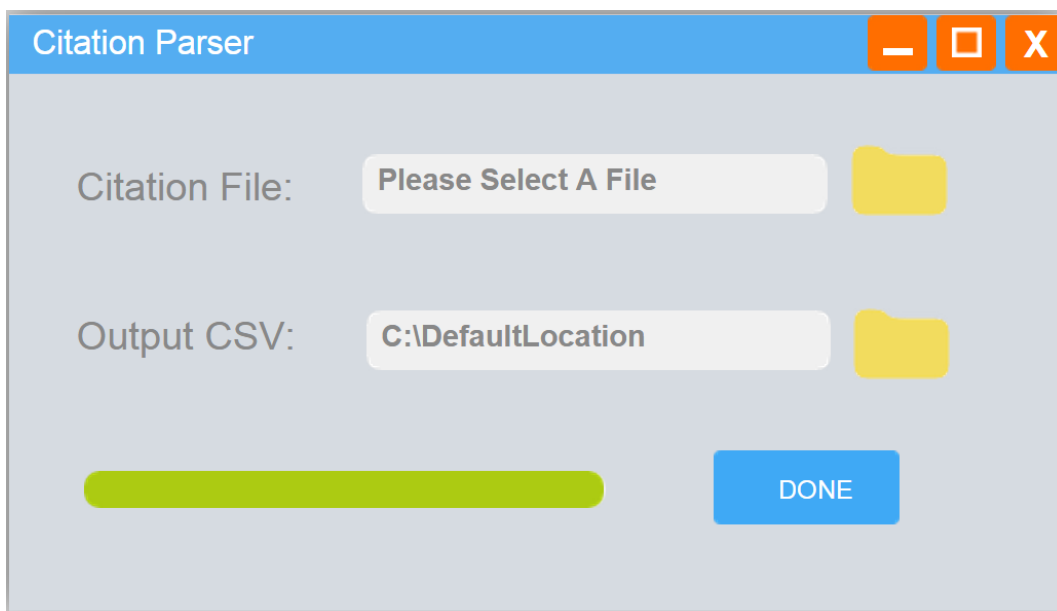
Below is our the idea for a flow model of the user interface. **Figure 5a.** shows the main screen where you select which file you want to parse, and where you want the output to go. **Figure 5b.** shows an example of the progress bar once the parsing has begun, and allows it to be cancelled.. **Figure 5c.** shows an example of the progress bar being completed for once the parsing program has finished. In these images it still just has the placeholders for the files, but once being used it would have actual filenames/destinations. **Figure 5d.** shows an example of what would happen if the parsing program encountered a problem. It gives a pop-up message telling the user where the output log will be.



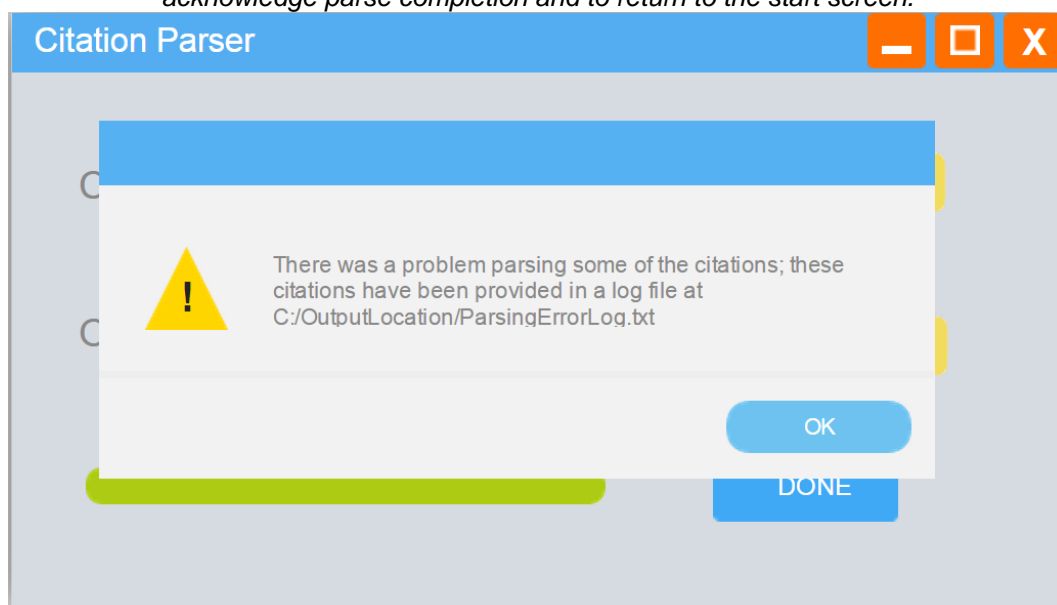
**Figure 5a.** Shown above is the start screen for the Citation Parser. This screen allows for selection of input/output file location and starts the parsing program.



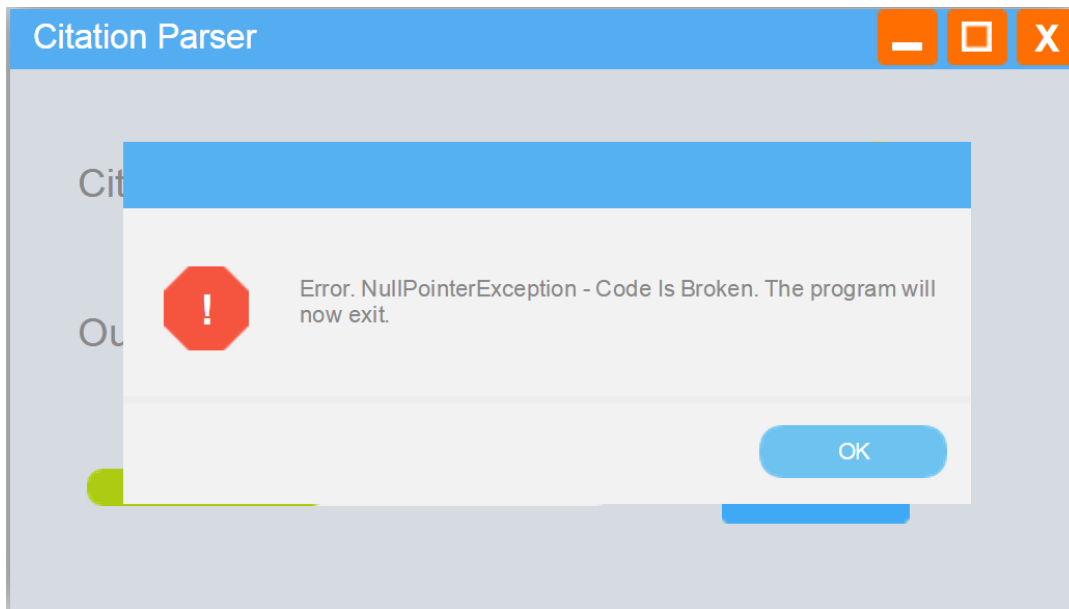
**Figure 5b.** Shown above is the in progress screen for the Citation Parser. This screen displays the current progress of the program and allows cancellation of the parsing.



**Figure 5c.** Shown above is the completion screen for the Citation Parser. This screen allows the user to acknowledge parse completion and to return to the start screen.



**Figure 5d.** Shown above is the warning pop-up for the Citation Parser. The warning alerts the user to an log file created when some citations can not be parsed.



**Figure 5e.** Shown above is the error pop-up for the Citation Parser. The Error Alert will open whenever there is an error reading/writing files, or if the program critically fails.

## 4) Implementation Plan

### 4.1) Programming Language

The client requested that the final program be runnable on Windows, Mac, and Linux so we need a language capable of running on each of these operating systems; to meet the given criteria, we selected Java as our programming language for the project. Java is also an object-oriented language, which is ideal for the project, since we are breaking up the citations into “objects” for our parsing logic. The GUI is going to be created using JavaFX<sup>3</sup>, which allows simple, attractive interfaces to be created cleanly and easily.

### 4.2) Tools Employed

We are utilizing the latest version of Eclipse<sup>4</sup> as our IDE. It comes with several handy features such as a visual debugger, and a GIT plugin. Since we are all going to be working on the same program, it will be useful to have the ability to grab the latest branches and merge changes all from within Eclipse<sup>4</sup>.



Here you can see from inside Eclipse<sup>4</sup> that we can view local and remote branches that are currently being tracked.

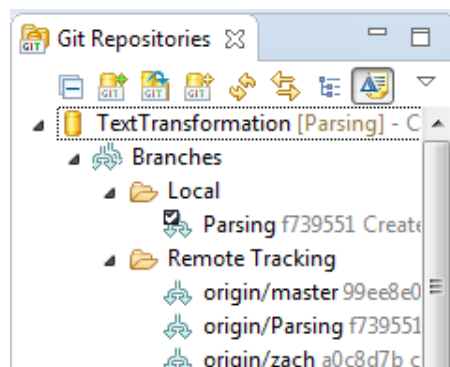


Figure 6. Eclipse<sup>4</sup> built in source control.

### 4.3) Parsing Strategy

Our parsing strategy for the citation file will revolve around several combinations of regular expressions to break up the citations and identify the proper parts. To make adding more regular expressions easy, we will develop a separate tool which builds up a tree of regular expressions. The top of nodes of the tree are very general regexes and as you go to the leaf nodes in the tree they become more specific. Our program will use the tree to find which regex matches each citation and it will traverse the tree getting more and more specific until it has an exact match. The purpose of using a tree is to save time, because now it won't have to go through every possible regex.

### 4.4) Source Control

We are using GIT for our code's source control. It allows us to keep track of what changes we make, and who is making them. It also provides the functionality to revert to a previous commit if something were to go wrong, and we need to return to an earlier point. We have a master branch, and then we have other branches for specific purposes (such as parsing, GUI, etc) which we can work on and make changes to. Once a branch gets to a solid state and after a code review then will we merge with the master branch. See **Code Review** section for details on our code review process. Here you can see an example of a source file in the master branch on Github<sup>5</sup>.



Figure 7. Github<sup>5</sup> image example

#### 4.5) Agile Process Tracking

To track our progress and tasks to do we are using an online scrum board at Trello.com. Our columns consist of backlog, to-do, in-progress, test, code review, and completed. The backlog will consist of cards that are stretch goals that we might not have time to get to. To-do are tasks that are waiting to be worked on. In-progress are our tasks that we are currently implementing. The test column is for cards that need to be tested. Code review is for cards that have been tested and need to be reviewed before merging with the master branch. See the **Code Review** section for more details on our code review process. Once the task has been merged to the master branch then it is finally completed and will be moved to the completed section.

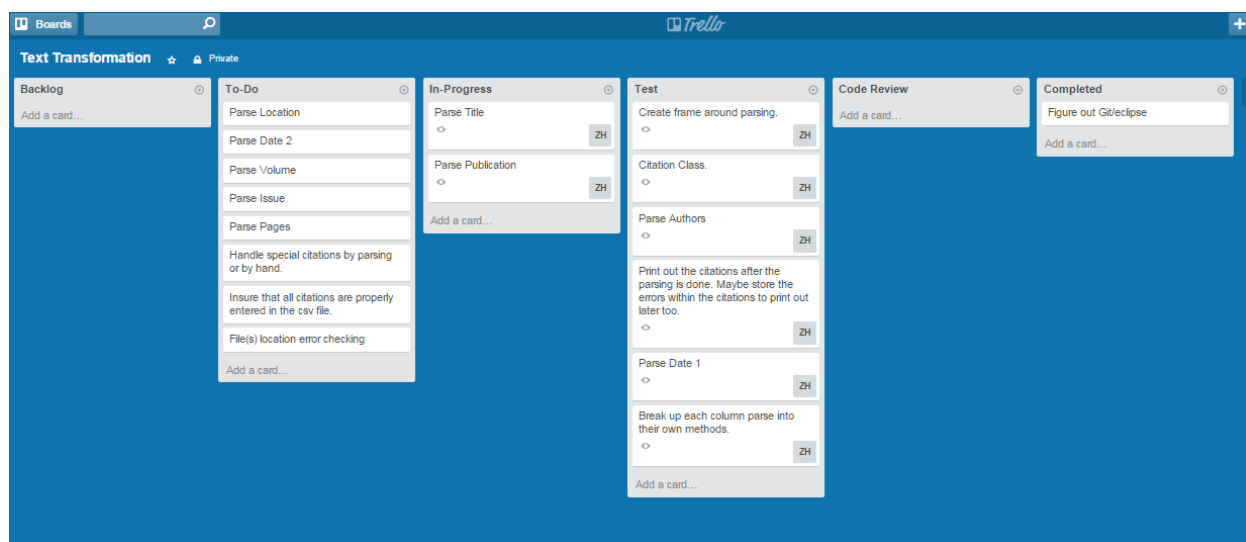


Figure 8. Shown above is an image of our scrum board early on in our development.

#### 4.6) Code Review

In order to assure that our code works properly, is clean, readable, and easily maintainable, we are enforcing code reviews of up to two people before

commits are pushed and merged into the master code branch. Code reviews will allow for other team member input on code and get constructive feedback and criticism back to the code author so that we can produce the best possible product. At least 2 group members must review major changes before merging. For smaller changes, 1 person would be okay, and for incredibly miniscule changes; no review would be necessary.

#### 4.7) Timeline

- Feb. 9-23
  - Finish the wiki write up and get it signed by our client and Dr. Fox. (Completed)
  - Setup code repositories and tools required for development(Completed).
- Feb. 23-March 9
  - Put Requirements and Design into Project Report. (Completed)
  - Determine tools and libraries to use. (Completed)
- March 9-March 23
  - Set up an initial user interface.
  - Set up testing classes and initial tests for the utility. (Completed)
  - Deliver an initial working demonstration to the client by March 20th.
- March 23-April 6
  - Add in more parsing utility code to improve citation coverage.
  - Add error output code to parser.
  - Add error output to user interface.
  - Increase code coverage of tests.
- April 6-April 20
  - Finalize project for client.
    - Demonstrate project for client.
    - Hand over source code, documentation, and compiled executable.
  - Final push for improved performance.
  - Assure that the project will run on different platforms as expected.

#### 4.8) Roles

We have three main coding roles and three external roles in this project. The roles are:

- **Backend Developer:** The backend will undoubtedly be the bulk of the project; it is, after all, the backbone. Without it, we do not have a project. Therefore, it is imperative that every team member collaborates and contributes to it. We will all need to understand the inner workings of the project in order to explain it, test it, develop an interface for it, and, ultimately, extend it. The backend will include overall design, the parsing structure, and performance concerns.

- **GUI Developer:** The GUI will be a small but important part of the utility. As Dustin is the most experienced with the JavaFX<sup>3</sup> and Java Swing utilities, he will be in charge of designing and deploying the interface. The interface will be responsible for file input and output selection, passing this information to the backend, and displaying any errors that arise when using the tool.
- **Test Developer:** Unit testing will be a critical and large part of ensuring that the utility acts as expected and does not collapse under unexpected circumstances. While it is large, as it will be developed in concert with the backend to ensure correctness as we proceed, only one person will be necessary to develop the tests. Fenton is experienced with testing through work experience and therefore will be in charge.
- **Point of Contact:** It will be a simple job, but a necessary one for the project. Fenton has been designated for contacting the client, the VTTI<sup>2</sup>, and the professor, in order to obtain meeting times, updates, and send information to any interested parties.
- **Scrum-Master:** Zach person is in charge on making sure everybody is using the scrum board correctly and that cards are moving across the board. Also, he is in charge of making sure the Github<sup>5</sup> branches are up-to-date, deleted, etc.
- **Presentation Manager:** Kevin is in charge of making sure that the presentation is finished and of informing the group about who is presenting what.

## 5) Prototyping

### 5.1) Regular Expression Tree

In order to save processing time and make it easier to branch out regular expressions so that our text transformation project can quickly identify the best solution to a problem, we use a tree structure that the transformation project traverses. Each node of the tree contains the following information:

- A **data** field, which contains the compiled regular expression we want to use;
- A **children** field, which contains all instances of children of this node;
- An **order** list, which contains a list of Order enumerations dictating the order of the string, for when we piece the regular expression matches back together.

This tree is compiled from binary data created by secondary tool we have developed to allow for ease of creating a tree structure. The secondary tool and binary data structure are detailed below.

### 5.2) Tree Tool

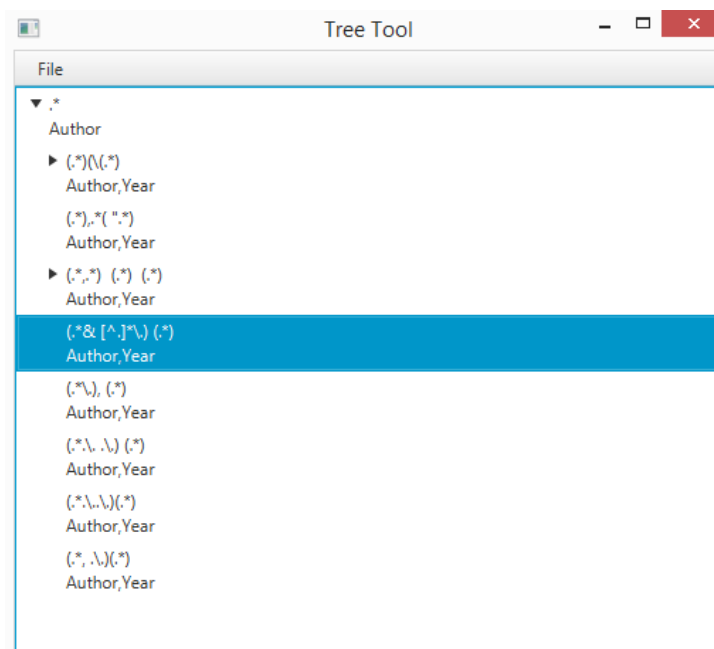
The Tree Tool was developed so that the team could easily build and edit a Regular Expression Tree as development moved forward. The tool allows developers to create a tree from scratch or to import an existing tree from a binary file; when the developers are done, they can export the tree on screen into a binary file format.

To edit any node in the tree, double click the node and edit the text in the text boxes that appear. The top text field should contain only Java compliant regex strings, and the bottom text field should contain citation orderings separated by commas. Acceptable citation orderings are:

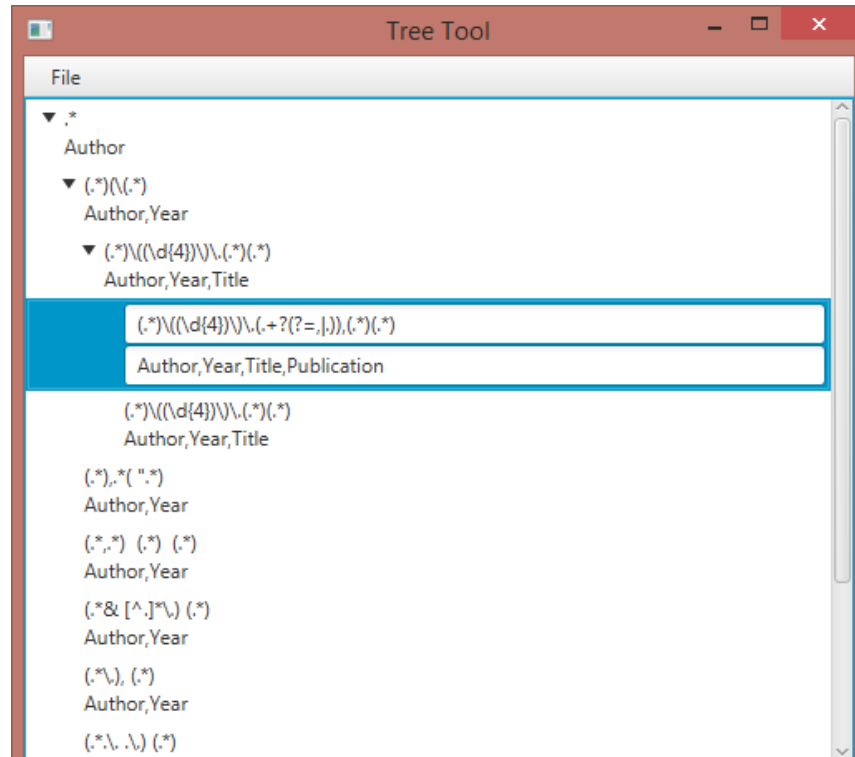
- Author
- Year
- Title
- Publication
- Location
- Date
- Volume
- Issue
- Pages

Moving forward, the team plans to implement some updates to the Tree Tool; however, as the Tree Tool is a secondary developer tool, it does not take top priority. Possible updates include:

- Improved GUI, using CSS
- Text Indicating text field purpose
- Integration with the main Citation Parser Tool



**Figure 6a.** The Tree Tool used to create Regex Parsing Tree Files. This image shows an example of a Regex Tree being imported and edited in the tool.



**Figure 6b.** This image shows the process for editing the regex and order fields in the tree.

### 5.3) Binary Data Structure

The binary data structure was developed in order to allow easy transfer of information from the Tree Tool to the Text Transformation utility, as well as easy modification within the Tree Tool. Within the binary, there are two chunks of information:

- The **String Table**;
- The **Node Object**.

The chunks are composed as follows:

#### i) String Table

The string table has a header of 32 bytes, composed of:

- 4 bytes, the **identifier**; always "STbl";
- 4 bytes, the **size of the chunk**, disregarding the identifier and size;
- 8 bytes, **zeroed**, for future-proofing;
- 4 bytes, the **number of entries** in the string table;
- 4 bytes, **zeroed**, for future-proofing;

- 4 bytes, the size of the **string table**;
- 4 bytes, the size **leading up to the string table**, including these 4 bytes.

```
53 54 62 6c 00 00 01 b1 00 00 00 00 00 00 00 00 STb1...±.....
00 00 00 0c 00 00 00 00 00 00 00 d9 00 00 00 c4 .....Û...[
```

**Figure 7a.** The hexadecimal representation of the header of our string table chunk.

Next, we have the string table entry section:

- 8 bytes, the **string key**;
- 4 bytes, the **offset to the string for the key**, starting at the beginning of the string table;
- 4 bytes, **zeroed**, for future-proofing.

```
d2 f8 55 48 2a 5e e6 d0 00 00 00 00 00 00 00 00 ÔøUH*^æÐ.....
4e f1 e9 34 9a 0c 65 d8 00 00 00 03 00 00 00 00 Nñé4š.e0.....
15 dc ad 6f 38 20 78 a6 00 00 00 0e 00 00 00 00 .Û-o8 x!.....
```

**Figure 7b.** The hexadecimal representation of three string table entries.

Finally, once all of the entries are done, we have our string table- a dump of strings, each separated by a zeroed byte.

```
2e 2a 00 28 2e 2a 29 28 5c 28 2e 2a 29 00 28 2e .*.(.*) (\(.*) .(.
2a 29 2c 2e 2a 28 20 22 2e 2a 29 00 28 2e 2a 2c *),.*(".*).(.*,
2e 2a 29 20 20 28 2e 2a 29 20 20 28 2e 2a 29 00 .*) (.*) (.*).
```

**Figure 7c.** The hexadecimal representation of our string dump.

When the string table is parsed, each program runs through the chunk, grabs all the entries, gets the string from the offset, and adds it to a key-value list. The object then uses the key-value list to find the matching regular expression from string keys contained within its binary.

## ii) Node Object Chunk

The node object chunk has a header of 28 bytes, composed of:

- 4 bytes, the **identifier**, always “**NOBJ**”;
- 4 bytes, the **size of the chunk**, disregarding the identifier and size;
- 8 bytes, **zeroed**, for future-proofing;
- 4 bytes, the **number of nodes** to read;
- 4 bytes, the **size of a node entry**;
- 4 bytes, **zeroed**, for future-proofing.

```
4e 4f 42 4a 00 00 03 50 00 00 00 00 00 00 00 00 NOBJ...P.....
00 00 00 0c 00 00 00 3c 00 00 00 00 .. ..<.....
```

**Figure 7d.** The hexadecimal representation of the header of our node object chunk.

Next, we have the node entry section:

- 8 bytes, the **string key** containing the regular expression data in the string table section;
- 4 bytes, **zeroed**, for future-proofing;
- 4 bytes, the **number of children** for this node;
- 4 bytes, the **offset** to the children of this node;
- 4 bytes, the **number of Order elements** for this node;
- 4 bytes, the **offset to Order data** for this node;
- 32 bytes, **zeroed**, for future-proofing.

String tables are **always** parsed first due to the necessity of finding string keys for regular expression data. The nodes are stored in breadth-first order; this is due to the fact that node children need to be stored next to each other so that any parser can run recursively through the file to import the tree. After all nodes, there is a raw dump of **Order** data.

**Figure 7e.** The hexadecimal representation of a node entry.

The order data is stored as follows:

- 4 bytes, the **enumeration ordinal**.

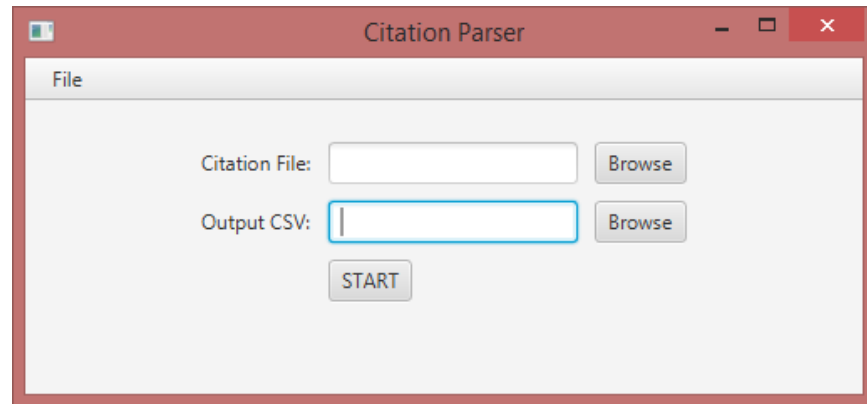
**Figure 7f.** The hexadecimal representation of ordinal dump data.

For example, ordinal 1 represents **Author**, ordinal 2 represents **Year**, and so on.

The binary data structure was designed so that the size of an entry could be easily modified without changing too much code internally, yet could feasibly store as many children and order elements as the byte limit allowed (4 bytes being an integer). There can be even more string keys stored as the type of key is a long integer, meaning that for most projects, the node binary format should suffice and perform well.

## 5.4) Citation Parser Tool





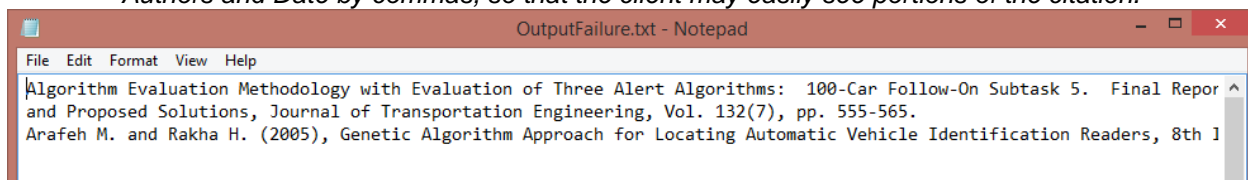
**Figure 7.** The Citation Parser Tool. This is the tool used by the client to give an input text file with citations in it, and to select a destination for Output files to be saved in.

The primary tool developed for the client is the Citation Parser Tool; this tool was created using JavaFX<sup>3</sup> libraries and for prototyping purposes, only runs from the developers IDE. The Parser Tool is responsible for and currently capable of the following:

- Reading an input text file that contains a single citation string on each line
- Importing Regex Tree Binary Files to use as the basis for parsing citations
- Parsing citation strings into an output CSV file
- Creating an Error file that contains citations that were not able to be parsed

Aatique, M.   Mizusawa, G.   Woerner, B.	1997	Performance of hyperbolic position location techniques for code division multiple access
Ahmadian, M.	1997	A hybrid semiactive control for secondary suspension applications
Ahmadian, M.	1997	Semiactive control of multiple degree of freedom systems
Ahmadian, M.	1999	Design and Development of Magneto-Rheological Dampers for Bicycle Suspensions"
Ahmadian, M.	1999	Filtering Effects of Mid-Cord Offset Measurements on Track Geometry Data"
Ahmadian, M.	2000	Application of Magneto-Rheological Dampers for Controlling Fire Out of Battery for Heavy Weapons. United Defense
Ahmadian, M.	2001	An Evaluation of the Practical Design Aspects of Magneto-Rheological Dampers for Controlling Gun Recoil Dynamics. United Defense
Ahmadian, M.	2001	On the design of magneto rheological recoil dampers for fire out of battery control. Proceedings of the 10th Gun Dynamics Symposium
Ahmadian, M.	2001	The noise and vibration benefits of soft-mounted locomotive cabs. Proceedings of 2001 IEEE/ASME Joint Rail Conference
Ahmadian, M.   Ahn, Y.K.	2000	Performance Analysis of Magneto-Rheological Mounts"
Ahmadian, M.   Cantwell, R.	1999	Effect of Pedestal Liner Clearance and Resilience on Rail Vehicle Operation"
Ahmadian, M.   DeGuilio, A.P.	2001	Recent advances in the use of piezoceramics for vibration suppression. Shock and Vibration Digest
Ahmadian, M.   Eisenhower, B.A.	1997	Application of Magneto-Rheological Suspensions to Intelligent Transportation Systems"
Ahmadian, M.   Glass, J.E.	2001	A Comprehensive Experimental Analysis of Kinematics and Dynamics of Volvo Optimized Air Suspension – 2. Volvo Trucks North America.
Ahmadian, M.   Huang.		
Ahmadian, M.   Huang, W.	2001	An Analytical Analysis of Kinematics and Dynamics of Volvo Optimized Air Suspension – 2. Volvo Trucks North America.
Ahmadian, M.   Huang, W.	2001	An Analytical Investigation of Power Hop in Heavy-axle Vehicles. Final Report
Ahmadian, M.   Huang, W.	2002	A Qualitative Analysis of the Dynamics of Self-Steering Locomotive Trucks"
Ahmadian, M.   Jeric, K. M.	1999	The Application of Piezoceramics for Reducing Noise and Vibrations in Vehicle Structures"

**Figure 8a.** The output file from the citation parser. This is a CSV file that separates sections such as Authors and Date by commas, so that the client may easily see portions of the citation.



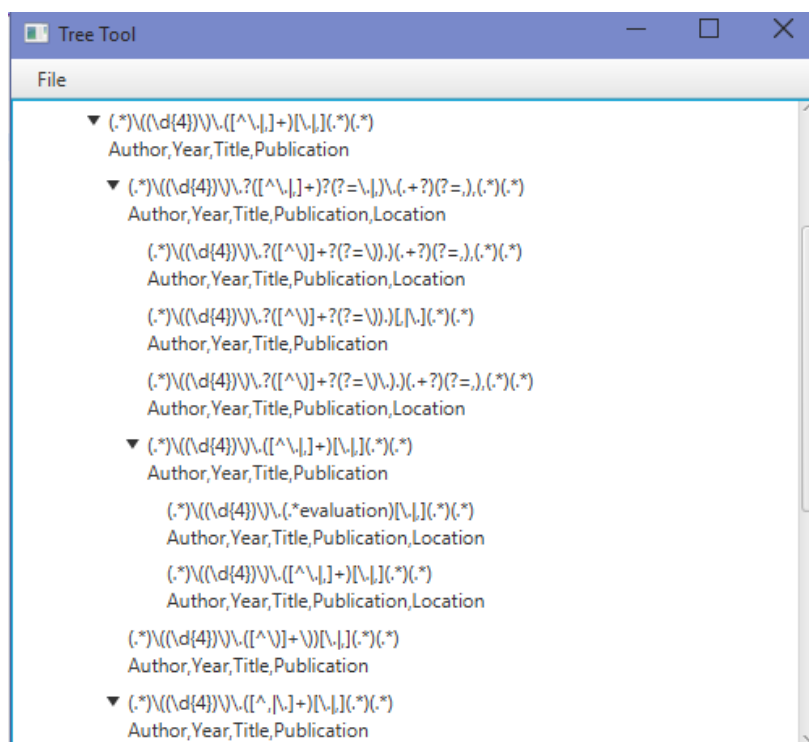
**Figure 8b.** The Error Output File. This file contains all citations or text that failed the parsing process.

Moving forward the tool will be extended so that it supports the following:

- Running as an executable Jar
- Using a complete, developer made Regex Parsing File by default
- Enhanced UI using CSS color and formatting
- Warning Dialog for when a citation is not able to be parsed
- Error Dialog and Error handling for program errors
- Possible addition of a progress bar, if needed for slower computers

### 5.5) Refinement 1

Our prototyping phase left us much further ahead than we anticipated; therefore, refinement has been relegated to simply improving the tree structure and tying the user interface into the backend code, making improvement and fixing bugs where need be. Our first iteration of the tree is visualized as follows:



**Figure 9.** *The Refined Tree. This tree parses over 1,000 citations, of which over 90% are correct.*

This refined tree parses out the authors, year, and titles of over 1,000 citations, and the rest are filtered to nodes that will be appropriately filled at a later date. Work remains to parse the rest of the strings, and it is anticipated that the tree will grow in size appropriately.

Further refinement going forward will still involve increasing the citation coverage of the regex tree, but will also involve debugging and fine tuning the GUI. A GUI change we are planning to do is instead of having the binary tree file

input in a drop down we are going to add another field like the citation file and output location. We are also planning to make the GUI more informative to the user by adding some descriptions and tips.

	A	B	C	D	E	F
	Author	Year	Title	Publication	Location	Date
1	Aatique, M.   Mizusawa, G.   Woerner, B.	1997	Performance of hyperbolic position location techniques for code division multip	Proceedings of Wireless '97, Calgary, Canada, July 9-11, 1997.		
2	Ahmadian, M.	1997	A hybrid semiactive control for secondary suspension applications	Proceedings of the Sixth ASME Symposium on Advanced Automotive Technologies,		
3	Ahmadian, M.	1997	Semiactive control of multiple degree of freedom systems	Proceedings of 1997 ASME Design Engineering Technical Conference, Sacramento, C		
4	Ahmadian, M.	1999	Design and Development of Magneto-Rheological Dampers for Bicycle Suspensi	Proceedings of 1999 ASME International Congress and Exposition, Nashville, Tennes		
5	Ahmadian, M.	1999	Filtering Effects of Mid-Cord Offset Measurements on Track Geometry Data	Proceedings of 1999 IEEE/ASME Joint Rail Conference, Dallas, Texas.		
6	Ahmadian, M.	2000	Application of Magneto-Rheological Dampers for Controlling Fire Out of Battery	United Defense, L. P.		
7	Ahmadian, M.	2001	An Evaluation of the Practical Design Aspects of Magneto-Rheological Dampers f	United Defense, L. P.		
8	Ahmadian, M.	2001	On the design of magneto rheological recoil dampers for fire out of battery conti	Proceedings of the 10th Gun Dynamics Symposium, Dallas, TX.		
9	Ahmadian, M.	2001	The noise and vibration benefits of soft-mounted locomotive cabs	Proceedings of 2001 IEEE/ASME Joint Rail Conference, Toronto, Ontario.		
10	Ahmadian, M.   Ahn, Y.K.	2000	Performance Analysis of Magneto-Rheological Mounts	Journal of Intelligent Material Systems and Structures, Vol. 10, No. 3, March, pp. 248.		
11	Ahmadian, M.   Cantwell, R.	1999	Effect of Pedestal Liner Clearance and Resilience on Rail Vehicle Operation	Proceedings of ASME Rail Transportation Division, Chicago, Illinois.		
12	Ahmadian, M.   DeGiulio, A.P.	2001	Recent advances in the use of piezoceramics for vibration suppression	Shock and Vibration Digest, Vol. 33, No. 1, 15-22.		
13	Ahmadian, M.   Eisenhower, B.A.	1997	Application of Magneto-Rheological Suspensions to Intelligent Transportation S	Final Report, Center for Transportation Research, October 1997.		
14	Ahmadian, M.   Glass, J.E.	2001	A Comprehensive Experimental Analysis of Kinematics and Dynamics of Volvo O	Volvo Trucks North America.		
15	Ahmadian, M.   Huang, W.	2001	An Analytical Analysis of Kinematics and Dynamics of Volvo Optimized Air Suspe	Volvo Trucks North America.		
16	Ahmadian, M.   Huang, W.	2001	An Analytical Investigation of Power Hop in Heavy-axle Vehicles	Final Report, Goodyear Tire and Rubber Company.		
17						

**Figure 10.** Example of some of the citations from our output file given the VTTL\_pubs.txt as input

You can see that up to the location field the citations are parsed correctly. As of now the output is creating 1011 parsed citations out of around 1456 citations given. Future iterations of our code will refine the parsing of author, year, title, and publication in order to increase the total citations parsed. From there we will move on to location, date, volume, issue, and page.

## 5.6) Refinement 2

Refinements this week were based solely on the merits of the tree. Expansions were made to ensure that as many Authors, Years, and Titles of citations were being parsed out as possible. Below are examples of both the expanded tree structure and the resulting output, showing the improved correctness of our parsing engine.



**Figure 14.** Sample of CSV output when running the Parser with the Refined RegexTree  
**Figure 15.** Example of some of the citations from our failed file given the VTTI\_pubs.txt as input

Figure 11 shows us a sample of the failed output file, which shows us which citations don't get matched to one of the nodes in our regex trees. We will use this file to create more specific regular expressions to cover more citations. Viewing the citations in this file we can determine what our tree is missing. As we add more regex nodes into our tree, this output file will shrink, until eventually we reach the goal of hitting all or most of the citations. For any that we absolutely cannot hit with the tree, we can manually add into the ending csv file.

## 6) Testing

In order to assure that we have created the system we want, that the parts operate sufficiently independently of each other, and that the interface satisfies both the needs of the project and the usability requirements of the client, we have implemented both software-level testing and manual testing. They ensure correctness and expose some issues present in the code.

It should be noted that no code is perfect and different execution environments can bring about unexpected issues.

### 6.1) Software-based Unit Testing



```

public class TreeParserTests {

    private TreeParser parser;

    private String TEST_PATH = "TestFiles/TestTree.rgxt";
    private String TEST_PATH_WITH_CHILDREN = "TestFiles/TestTree_WithChildren.rgxt";

    private long TEST_KEY = 5814361054088246626L;
    private long FAIL_KEY = 100L;

    @Before
    public void setUp() throws Exception
    {
        parser = new TreeParser();
    }

    /**
     * Tests loading the tree into binary.
     * @throws IOException
     */
    @Test
    public void TestVerifyFileExists() throws IOException {
        byte[] result = parser.loadFile("");
        Assert.assertNull(result);
        result = parser.loadFile(TEST_PATH);
        Assert.assertNotNull(result);
    }
}

```

**Figure 16.** Demonstration of code-coverage plugin, indicating that the code shown is hit in-test.

Programmatically ensuring that code is working is an important part of software development; being able to easily run tests, change code, re-run, and assert that execution proceeds as expected is crucial to delivering a top-notch product.

In order to assure the best for this project, we used a third-party tool called **EclEmma**<sup>6</sup>. When this tool is used in Eclipse<sup>4</sup> (our development environment), it runs all unit tests, and displays in code where code is either:

- Hit
- Hit some branches of for loops, switch cases, etc.
- Not hit at all.

This tool assisted in developing tests for the backend system - the text/file parsers, and the transformation into citations. The front end of the system was tested manually.

To test the back end of the system, four files were created:

- A tree containing one single node.
- A tree containing one single node, and a child.
- A tree containing one matching regular expression.
- A text file with four text citations:
  - One that passed the tree
  - Three that failed the tree for various reasons.

These files, in conjunction with unit tests that simulated the interface, successfully tested important elements of execution such as failed key-value lookup, failed file import, asserting the correct number of nodes, and more.



## 6.2) Manual Interface Testing

Our user interface across both of our tools were simple enough to manually test, and we deemed it unnecessary to take the extra time to create GUI hooks for testing; rather, we would manually verify that the interfaces worked as expected, and make changes based on what we thought were appropriate.

For the Tree Tool, testing helped us to determine several areas of incorrectness:

- An empty area at the bottom of the screen that could not be resized
- Sometimes, clicked nodes opened for edit would display text that was not associated with that node.

We were able to successfully replicate and eliminate these issues through manual testing.

For the Text Transformation tool, we determined that the only major issue we were running into was the fact that the “Import Binary” dropdown was more appropriately placed as another input box, similar to the boxes indicating the file locations for text input and CSV output.

The Text Transformation tool’s GUI backend interfacing with the actual transformation was sufficiently simulated in the Text Transformation software-based unit testing.

## **Lessons Learned:**

### **1) Problems Encountered**

The main problem we ran into was that there were so many different variations in the citations that our parsing program really wasn’t able to achieve 100% coverage. These variations stem from the fact that this list of citations we were given isn’t complete, so some citations are incorrect and missing different parts. It will require someone to manually go through to add in the citations we were unable to hit.

Another problem we faced was that sometimes the parsing program’s output isn’t perfect. It has around 80% success rate. Sometimes though it has an error and ends up sticking parts of the citations in the wrong fields. This is because of situations where it is very difficult/near impossible to tell a citation’s title and publication apart from each other. Fixing these errors in the output will require some manual labor to fix.

### **2) Future Work**

This project has a lot of potential for being used in the future. We designed it with extensibility in mind. The developer tree tool allows someone to create a regex tree

tailored specifically to what they want to parse. With some development work, this parsing program could easily be made to parse other institutions citation lists as well. Since it is so customizable, it can search for whatever the client needs it to.

## **Acknowledgements:**

1. We would like to thank our client, Nathan Hall. He met with us often and was very clear and helpful when communicating what he wanted from us. He was very understanding with the issues we had and adjusted what he expected from the project as we went along. Through him we also got a meeting with some people from the VTTI<sup>2</sup> which was a valuable experience to see how a real-world client meeting would go.
2. We would like to thank our professor Dr. Fox. He was able to give us some tips with the project as well as giving us feedback along the way. He also answered questions we had about deadlines and such.

## **References:**

1. Java
  - a. <https://www.java.com/en/>
2. Virginia Tech Transportation Institute (VTTI)



- a. <https://www.vtti.vt.edu/>
  - b. Copyright © 2015 Virginia Polytechnic Institute and State University
- 3. JavaFX
  - a. <http://www.oracle.com/technetwork/java/javase/overview/javafx-overview-2158620.html>
- 4. Eclipse
  - a. <https://eclipse.org/>
  - b. Copyright © 2015 The Eclipse Foundation.
- 5. Github
  - a. <https://github.com/>
  - b. Copyright © 2015 GitHub, Inc.
- 6. EcLemma
  - a. <http://www.eclemma.org/>
  - b. Copyright © 2006, 2015 Mountainminds GmbH & Co. KG and Contributors
- 7. FreeCite
  - a. <http://freecite.library.brown.edu/>
  - b. Brought by Brown University (<http://www.brown.edu/>)
  - c. Implemented by Public Display (<http://www.pubdisplay.com/>)
  - d. Funded by the Mellon Foundation (<https://mellon.org/>).
- 8. Agile
  - a. <http://agilemethodology.org>