Hochschule für Technik
und Wirtschaft Berlin
University of Applied Sciences

BACHELOR THESIS

similarity texter: A text-comparison web tool based on the "sim_text" algorithm

BY
Sofia Kalaidopoulou

MATRICULATION No.
522789

SUBMISSION DATE
March 7, 2016

FACULTY
Internationale Medieninformatik
International Media and Computing

FIRST SUPERVISOR
Prof. Dr. Debora Weber-Wulff

SECOND SUPERVISOR
Prof. Dr. Barbara Kleinen

Abstract

Text comparison constitutes a major field of interest in computer science, given the vast spectrum of fields to which it can be applied. Information retrieval, document searching, plagiarism detection, machine translation, and DNA sequences detection, to name a few, are only possible due to the development of effective text similarity algorithms.

The present thesis discusses the *sim_text* algorithm for plagiarism detection, developed in 1989 by Dick Grune at the Vrije University of Amsterdam. Furthermore, it describes the implementation of a web tool, based on the aforementioned algorithm, that detects lexical similarities and highlights the optimal longest common substrings found between two input texts.

Table of Contents

List of Abbreviations

CSS	Cascading Style Sheets
DOCX	Microsoft Word 2007/2010/2013 document
flex	Fast Lexical Analyzer
GUI	graphical user interface
HTML	HyperText Markup Language
HTML5	fifth revision of the HTML standard
JS	JavaScript
LCS	longest common substring
MIME	Multipurpose Internet Mail Extensions
MTS	sequence of Min_Run_Size tokens
ODT	OpenDocument Text
PDF	Portable Document Format
SIM	software and text similarity tester
TXT	text file
UML	Unified Modeling Language
UX	user experience
XML	Extensible Markup Language

List of Algorithms

List of Figures

1 Introduction

Measuring lexical similarity still represents a subject matter for further research in modern computer science. Despite the large number of suggested string-based matching algorithms (Charras et al., 1997), none of them can provide the *absolute* answer to this problem: how to trace the maximum contiguous sequence of characters between two input strings, while keeping search time and space overhead as low as possible.

Integrated software solutions for plagiarism detection provide different approximations to the aforementioned problem, and therefore different degrees of efficiency (Weber-Wulff, 2014). In their vast majority, they are commercial desktop products, licensed under restrictive terms, which do not allow the disclosure of the source code, and require the possession of a license as a prerequisite for their use.

A free and open source desktop software solution, which has been tested, and proved to be efficient both in terms of plagiarism detection (Weber-Wulff, 2014) and of execution time, is the *software and text similarity tester* (*SIM*), developed by Dick Grune. However, due to its console-based implementation, this software is not intended for the average user.

The concept for the development of *similarity texter*, namely a free for non-commercial use, and open source web application, capable of measuring and highlighting the longest common substrings between two input files and/or texts, is justified by the following facts:

- An ever increasing demand for web applications is recorded, mainly due to their portability.
- Web-based applications of this kind are either scarce, inefficient, or not generally available for free use.

Given the proven efficiency of *SIM*, the implementation of the string matching algorithm in *similarity texter* is based on the same algorithm developed by Dick Grune for the purpose of tracing lexical similarities in natural language texts (i.e. *sim_text*

algorithm).

The present thesis is organized as follows:

Section 2 introduces the *software and text similarity tester,* developed by Dick Grune.
Section 3 describes the *sim_text* algorithm.
Section 4 discusses the implementation of *similarity texter*.

2 SIM: The software and text similarity tester

2.1 Overview

SIM is an open source program which was developed in 1989 by Dick Grune, a Dutch computer scientist and lecturer until 2005 at the Vrije University of Amsterdam. It measures, and reports lexical similarities in software projects as well as in natural language texts.

2.2 Background

SIM was originally developed as a tool for tracing similar stretches of code in large-scale software projects (e.g. compilers), so that they could be subsequently refactored into single modules (Grune et al., 1989). Given the good results that the *software similarity tester* delivered, Grune decided to use this program for the purpose of determining whether an assignment submitted by a student during a computer science workshop was a copy of somebody else's work. A manual detection of potentially plagiarized assignments was not feasible given the small number of supervisors as opposed to the large number of students attending the course. In his paper (Grune et al., 1989), he also underlines two major ethical problems for him as a lecturer that arise from situations of copied submissions: unfairness in terms of grading towards students who have not cheated, and failure of the educational process to fulfill its goal.

2.3 Supported languages and extendability

The current version (v2.89) of *SIM* provides checkers both for natural language text as well as for source code written in the following programming languages: C, Java, Pascal, Modula-2, Miranda, and Lisp. The program can easily be extended by including a lexical description of the new language.

2.4 Installation and execution

SIM is a command-line tool written in C that can be executed on all major operating systems: Linux, MS Windows, and OS X. Unfortunately, the developer does not provide out-of-the-box installers or compiled versions of the program for all systems. The only compiled version, available for download, is intended for MS Windows. Linux, and OS X users need to compile the C source files themselves in order to produce the corresponding executable binaries for their system. They may also need to install additional dependencies, e.g. *flex* (Fast Lexical Analyzer), before proceeding with compiling. A short description on the compilation process is given in the `README`, and `MakeFile` files that come together with the source code.

Compilation generates an equal number of executable files to the languages supported by the downloaded version. To run *SIM*, users must call the executable file that corresponds to the sort of files they want to compare (e.g. *sim_text* for text files, *sim_c* for c source files, etc).

2.5 Usability

SIM does not have a graphical user interface (GUI). Interaction with the program involves only the use of the terminal. However, by no means should it be assumed that the program is lacking in capabilities. On the contrary, it offers an extensive list of options that can parameterize its functionality. The ones that are relevant to *sim_text* are listed below. For clarity reasons, a categorization of the options is attempted, as follows:

General information options

`-M`	Displays information on the memory usage.
`-v`	Displays the program's version, and compilation date.

Input options

`-i`	Reads the file names from standard input.

`-R`	Reads the list of files in the specified directory recursively.

Output options

`-o <File>`	Writes results to the specified file.
`-w <Number>`	Sets the terminal's page width (in number of columns). Default value is 80.

Line-based

`-d`	Displays results in *diff* format.
`-n`	Summarizes results by file name, position, and size.
`-T`	Displays results in a more suitable format for post-processing.

Percentage-based

`-p`	Displays results as percentages.
`-P`	Displays results as percentages, showing only the main contributor.
`-t <Number>`	Suppresses output of results below the specified percentage threshold. Default value is 20.

Comparison options

`/` or "`|`"	Symbol for separating *old* from *new* files, e.g. `new / old`. *New* files are compared, and self-compared to each other. *Old* files are NOT compared or self-compared to each other.
`-e`	Compares each file to each other in isolation.
`-r <Number>`	Sets the minimum number of words for a match. Default value is 8.
`-s`	Suppresses self-comparison of a *new* file.
`-S`	Suppresses comparison between *new* files, including self-comparison.

2.6 Demonstration of use

The following call to the *sim_text* executable file compares the contents of the given two text files to each other, and outputs in a two-column formatted layout all similarities with a minimum length of two words. Self-comparison of input files

is suppressed.

Program call

```
sim_text -r2 -s file1.txt / file2.txt
```

Contents of `file1.txt`

```
To be, or not to be¶
```

Contents of `file2.txt`

```
To be and ¶
yet not ¶
to be¶
¶
```

Output

```
File file1.txt: 6 words, 1 line
File /: separator
File file2.txt: 7 words, 3 lines
Total: 13 words

file1.txt: line 1-1                  |file2.txt: line 2-3              [3]
To be, or not to be                  |yet not
                                     |to be


file1.txt: line 1-1                  |file2.txt: line 1-1              [2]
To be, or not to be                  |To be and
```

The above program call, i.e. arguments, and input files, will serve as an example (hereinafter referred to as the "model example") throughout the present thesis.

2.7 Conclusion

SIM is a solid, and powerful open source program that incorporates great capabilities. However, the absence of a graphical user interface, together with the lack of out-of-the-box installers for all platforms, may discourage average users from trying it.

3 SIM_TEXT: Unraveling the algorithm

This section discusses the *sim_text* algorithm, namely the algorithm for tracing similarities in natural language texts. It attempts to reveal, and document in a systematic manner the sequence of operations performed by the program, thus providing the reader with a solid understanding of the *logic* that lies behind the numerous lines of source code.

The following activity diagram offers a general overview on the way in which the program runs. It illustrates the order of execution of major routines during a single call to the *sim_text* executable file. For a more detailed overview, refer to the sequence diagram in Figure 21.

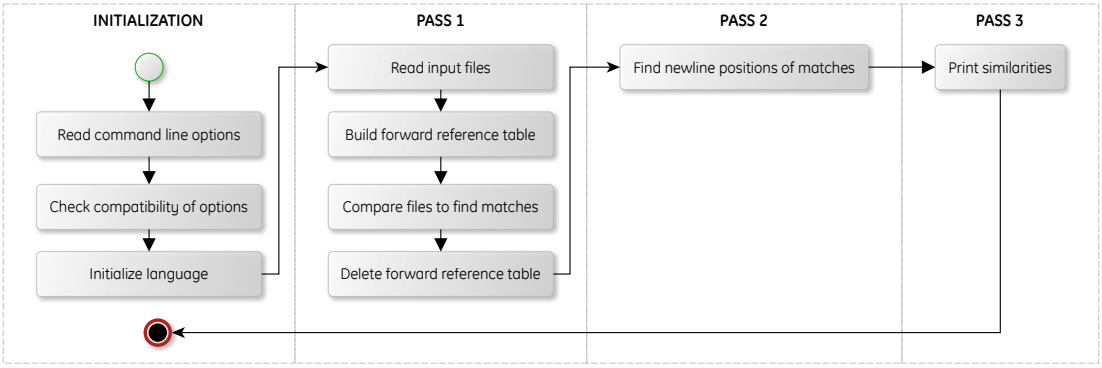

Figure 1. Basic activity diagram of *sim_text*

As can be seen, a call to the program involves the successful completion of four phases, namely initilization, pass 1, pass 2, and pass 3. Each one of these phases will be discussed separately in the subsections to follow.

3.1 Initialization

The program starts by reading in the parameters passed by the user; it checks their compatibility, and initializes global variables, such as the minimum number of words that constitute a match (`Min_Run_Size`), the page width (`Page_Width`), and/or the percentage threshold (`Threshold_Percentage`). It also initializes the appropriate

language, i.e. the grammar to be used for the file comparison. The last step is required for the right pre-computation of the tables for each grammar.

3.2 Pass 1

This phase, which constitutes the very essence of the *sim_text* algorithm, can be subdivided into the following two stages:

a) the *preprocessing* stage, which includes the reading and tokenization of the input files, as well as the construction of the forward reference table, and

b) the *searching* stage, which involves the actual comparison of the input files.

Each major step carried out during this phase, as shown in Figure 1, will be discussed in detail in the subsections to follow.

3.2.1 Reading the input files

During this step, each input file is read in sequence, and its contents are parsed, and split into lexical tokens, i.e. "words", as follows:

For the detection of meaningful "words" in the input, a lexical analysis is performed using an external lexical scanner, namely *flex*. The scanner reads the input on a character-by-character basis, and recognizes patterns as specified by the following rules, defined in the `textlang.l` file:

Definitions

Word element	Any letter of the Latin alphabet regardless of its case (`a-zA-Z`), any digit (`0-9`) or any character in the range of `128-255` of the extended ASCII table.
Non-word element	The inverse of a *word element*, i.e. all symbols other than the ones defined as a *word element*.

Rules

Newlines	
Tight word	A sequence of one or more *word elements*.
Spaced word	A sequence of one or more two-*word elements*, separated by one white space, and followed by one *non-word element*, e.g. "`i m!`".

Patterns that do not fall into the aforementioned rules are discarded.

Each lexical token returned by the scanner is then checked in order to be determined whether it constitutes a newline or a word (*tight* or *spaced*).

- If it is a newline, its position is stored in the array `nl_buff` as the difference between the lexical token counter's current position, and the last stored value in the array `nl_buff`. Figure 2 provides a better overview on the calculation of this value.
- If it is a word, all letters are converted to lowercase, and white spaces, if any, are discarded (in case of *spaced words*). The hash value of the word is then computed, and stored in the array `Token_Array`, starting at index 1. Algorithm 1 describes the hash function used for the conversion of a word into a hashed token.

Algorithm 1. idf_hashed(word)

Input : A sequence *word* of characters, representing a meaningful lexical token
Output: An integer h of type *unsigned short*, representing the *word*'s hash value

$h \leftarrow 0$ // 32-bit signed int
foreach *character* **in** *word* **do**
  $ch \leftarrow$ *character* & 0377
  if ch = *white space* **then**
    get next *character*
  $h \leftarrow (h \cdot 8209) + (ch \cdot 613)$
  if $h < 0$ **then**
    $h \leftarrow h + 2147483647$ // $2^{31} - 1$
if $h < 0$ **then**
  $h \leftarrow 0$
$h \leftarrow h \bmod ((1 \ll 16) - (1 \ll 9) - 1)$ // $0 \leqslant h < 2^{16} - 2^9 - 1$
$h \leftarrow h + (1 \ll 9)$ // $2^9 \leqslant h < 2^{16} - 1$
return (*unsigned short*)h

This algorithm converts a lexical token of an arbitrary length into a unique unsigned integer of 2-byte length. Left-bit shifting ensures that this integer fits entirely in the second byte, with its values ranging from 2^9 to $2^{16} - 1$ bits. The values of the first $2^9 - 1$ bits are reserved for particular identifiers, i.e. *simple tokens*, *summary tokens*, and *special tokens* (see comment in header file `token.h`).

For each file being read, a structure `text` is created, and stored in the array `Text`. Each structure records the name of the file, the index of its first `Token` (inclusive) and its last `Token` (non-inclusive) in the array `Token_Array`, the index of its first `Newline` (inclusive) and its last `Newline` (non-inclusive) in the array `nl_buff`, as well as whether it is terminated by a newline.

The following figure illustrates the above procedure for the given *model example*.

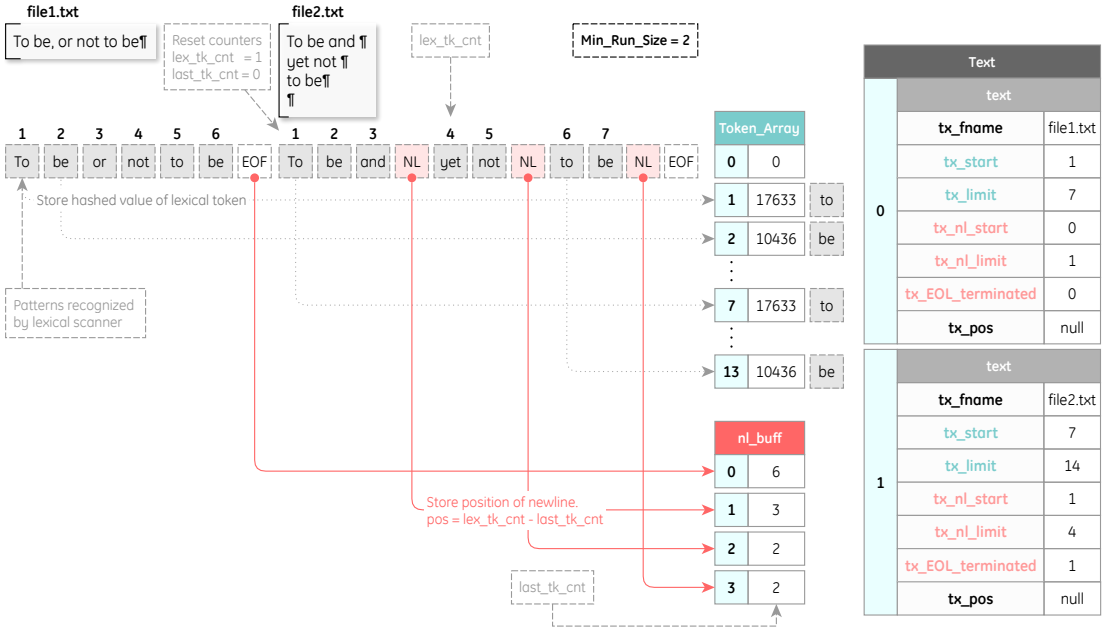

Figure 2. Graphical representation of the reading of input files

3.2.2 Building the forward reference table

During this step, the array `forward_reference` is constructed, which records the next position in the array `Token_Array`, at which a sequence of `Min_Run_Size` tokens (hereinafter referred to as "MTS") with the same hash value starts. The array is

initialized with a size that equals the next free index slot (`tk_free`) in the array `Token_Array`, i.e. the index of the last stored `Token` + 1.

To fill in this array, a hash table, namely `last_index`, is created; it stores at its index `h` the last occurrence, in the array `Token_Array`, of a *MTS* with a hash value `h`. For the initialization of the hash table, a prime number is used that has a value greater than that of `tk_free`. This number is drawn from the array `prime`, which defines a set of 27 primes in the form of $4 \cdot i + 3$ for some i, where $\frac{prime[i\text{-}1]}{2} < prime\,[i] < 2^{40}$.

The array `forward_reference` is populated as follows:
For each position `p`, except for the last `Min_Run_Size - 1` positions, in each `Text`, the hash value `h` of a *MTS* starting at position `p` is computed using Algorithm 2. If `last_index[h]` contains already a meaningful position, i.e. a value greater than 0, the current position `p` is written to the index `last_index[h]` of the array `forward_reference`. Finally, the value at index `h` of the hash table `last_index` gets updated with the current position `p`. Figure 18 provides a detailed diagram of the described algorithm. Figure 3 illustrates the creation of the array `forward_reference` for the given *model example*.

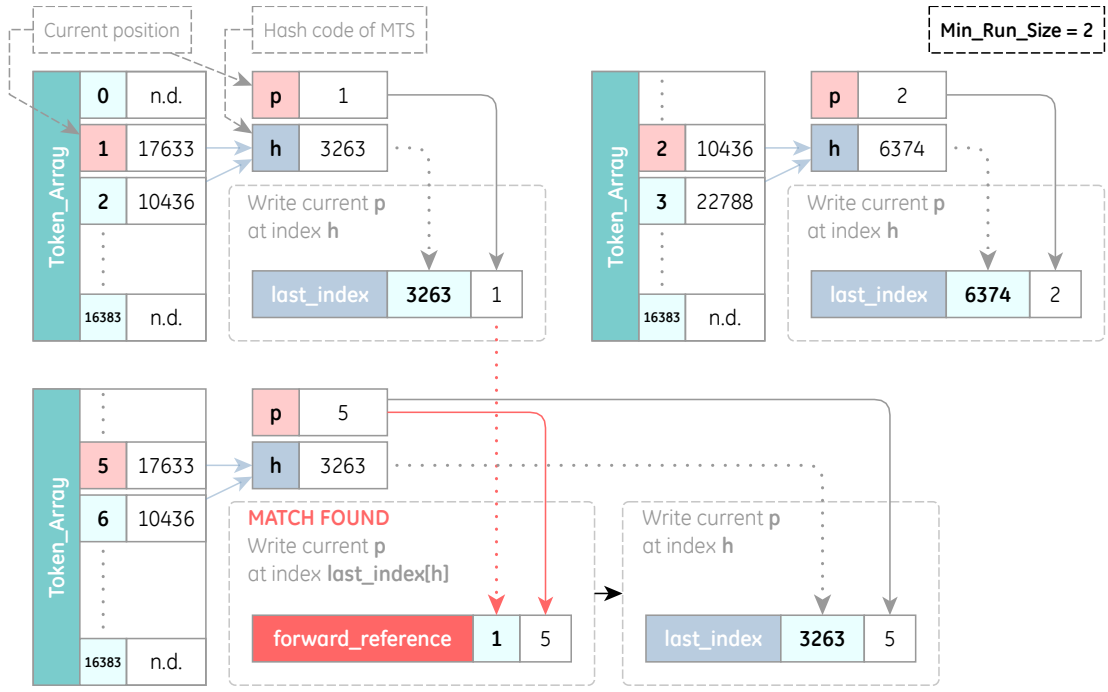

Figure 3. Stepwise graphical representation of the construction of the forward reference table

Algorithm 2 describes the implementation of the hash function, used for the creation of the hash table `last_index`. The function takes two arguments as input: a sequence `tokens` of an arbitrary number of positive integers of 2-byte length, and an array `sample_pos` of 24 positive integers of 4-byte length; it returns the hash code for the given sequence of integers. Computation involves left-bit shifting, which ensures that the value of the hash code remains within the range of $2^{31} - 1$ bits, as well as the application of the XOR operation to each sample position in sequence.

Algorithm 2. hash1(tokens, sample_pos)

Input : A sequence *tokens* of an arbitrary number of integers of type *unsigned short*, and an array *sample_pos* of *N_Samples* integers of type *unsigned*
Output: An integer h of type *unsigned*, representing the hash value of the sequence *tokens*

```
N_Samples ← 24
h ← 0                                              // 32-bit unsigned int
for n ← 0 to N_Samples − 1 do
    h ← h ≪ 1                                      // == h · 2
    if h & (1 ≪ 31) then                           // if left-most bit is 1
        h ← h^(1 ≪ 31 | 1)                         // move it to the end == h − 2^31 + 1
    h ← h^tokens[sample_pos[n]]
return h
```

One interesting aspect of the above implementation is the use of the array `sample_pos`. In order for the hash function to preserve the same running time during execution irrespective of the `Min_Run_Size` value, a sampling on the tokens is performed, thus reducing their number to 24. The positions of the sampled tokens are calculated only once for the given `Min_Run_Size` value, and stored in the array `sample_pos` as integers; the decimal part is discarded. The computation of each sample position is defined by the following formula:

$$f(n) \quad = \quad n \cdot \left(\frac{Min\_Run\_Size - 1}{N\_Samples - 1} \right) + \frac{N\_Samples - 1}{2 \cdot \left(N\_Samples - 1 \right)}$$

$$\text{where} \quad N\_Samples = 24 \quad \text{and} \quad 0 \leqslant n < N\_Samples$$

This approach has a positive effect in the case where the `Min_Run_Size` value is

greater than the number of `N_Samples`. In the opposite case, which is most likely to be for *sim_text*, the array `sample_pos` will contain duplicates.

At this point, the array `forward_reference` has already been created. The hash table `last_index` is of no need any more, and thus it is discarded. Figure 4 illustrates the created array `forward_reference` for the given *model example*.

To eliminate potential spurious references from the array `forward_reference`, which may be a common case especially when comparing large files, a second scanning is performed. This time the tokens are hashed using Algorithm 3, which returns a more representative hash code for a given *MTS*.

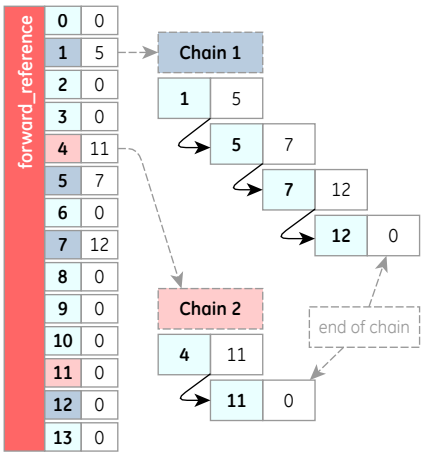

Figure 4. Graphical representation of the forward reference table

Algorithm 3. hash2(tokens, sample_pos)

Input : A sequence *tokens* of an arbitrary number of integers of type *unsigned short*, and an array *sample_pos* of *N_Samples* integers of type *unsigned*

Output: An integer h of type *unsigned long long*, representing the hash value of the sequence *tokens*

$N\_Samples \leftarrow 24$
$last\_pos \leftarrow N\_Samples - 1$ `// last_pos:= last index`
$h \leftarrow 0$ `// unsigned long long int`
`// 64:= size of unsigned long long (bits)`
$h \leftarrow h\hat{}((unsigned\ long\ long)tokens[sample\_pos[0]] \ll 0)$
$h \leftarrow h\hat{}((unsigned\ long\ long)tokens[sample\_pos[last\_pos]] \ll (64 \cdot 1/5))$
$h \leftarrow h\hat{}((unsigned\ long\ long)tokens[sample\_pos[last\_pos/2]] \ll (64 \cdot 2/5))$
$h \leftarrow h\hat{}((unsigned\ long\ long)tokens[sample\_pos[last\_pos \cdot 1/4]] \ll (64 \cdot 3/5))$
$h \leftarrow h\hat{}((unsigned\ long\ long)tokens[sample\_pos[last\_pos \cdot 3/4]] \ll (64 \cdot 4/5))$
return h

The elimination of spurious references reads as follows:
For each position `p`, except for the last `Min_Run_Size` positions, in the array `Token_Array`, the *MTS* starting at position `p`, and all referenced *MTS* belonging to the

same chain (`forwared_reference[p]`) are hashed using Algorithm 3. If their hash values match, the next position `p` is read; otherwise, references with non-matching hash values are overwritten, i.e. removed from the array `forward_reference`. Figure 18 provides a detailed diagram of the described algorithm.

3.2.3 Comparing the input files

During this step, the contents of the input files are compared to each other, and the longest common substrings of a minimum length, as defined by the `Min_Run_Size`, are recorded.

In *SIM*, the way in which the input texts are compared to each other is determined by the comparison options passed by the user during the program call. As stated in Subsection 2.5, input texts are divided into two categories: the *new* and the *old* ones. *Old* texts are never compared to themselves or to each other, whereas *new* texts may be compared to themselves as well as to each other, depending on which options are set. Consequently, one major task to be performed at the very beginning of the comparison process is to find out which input texts should be included in the comparison.

The **method `Compare_Files()`** in the file `compare.c` is responsible for carrying out this task. For each *new* text, the comparison options are read, and depending on their values, the indices of the first and the last target text to be compared to the current *new* source text are defined. Figure 19 provides a detailed overview of this routine.

Once the range of target texts that should be compared to each *new* source text is set, the actual comparison begins by a call to the **method `compare_one_text()`**. Each token that belongs to the current source text `txt0`, except for the last `Min_Run_Size` tokens, is compared to each token that belongs to the range of target texts, which start precisely at index `first` and end at index `limit - 1`. Comparison returns the longest common substring (LCS) of at least `Min_Run_Size` length. If a meaningful match is found, i.e. the value returned by the `lcs()` method is greater than 0, the match is recorded by a call to the method `add_run()`, and comparison in the source text resumes at token position *current token index+LCS size*. Otherwise, the next token

is compared. Figure 20 illustrates the `compare_one_text()` method.

As stated above, the **method `lcs()`** is responsible for recording the longest substrings, which are common in both source and target input texts. In this respect, the method makes use of the array `Forward_Reference` to speed up comparison as follows (see Figure 20 for a detailed overview of this routine):

Comparison in the source text starts at token position `i0`, and in the target text at token position `i1`, where `i0` and `i1` point to the same token. Each token in the source text is compared to each token in the target text, which belongs to the same *chain* of tokens, as defined by the forward reference table.

If the token at index `i1` does not belong to the range of the current target text(s), namely:

a) if the index `i1` is less than the index of the first token of the current target text `txt1`, comparison in the target text continues with the token at index `Forward_Reference(i1)`.

b) if the index `i1` is greater than or equal to the index of the last token of the current target text `txt1`, the current target text is skipped, and the next text which satisfies the above condition is assigned to `txt1`. Comparison goes on with the step described below.

If the token at index `i1` lies indeed within the range of the target text(s), a *backwards* comparison is attempted as follows:

If no best match has been recorded so far (i.e. first loop), comparison in the source text `txt0` starts at token position `i0 + Min_Run_Size - 1`, and in the target text `txt1` at token position `i1 + Min_Run_Size - 1`; this ensures that a match of at least `Min_Run_Size` will be tracked. Otherwise, comparison starts at token position `i0 + size_best`, and `i1 + size_best` respectively, thus ensuring the recording of the longest common substring between the last and the current comparison loop.

Starting precisely at the token positions as defined above, a sequence of tokens in the source text `txt0` is then compared to a sequence of tokens of corresponding size in the target text `txt1`. Each token of the source text's sequence is compared to

each token of the target text's sequence, starting from the last token and moving backwards to the first one.

- If the hash values of both sequences match, the size (`size_best`, i.e. the number of tokens with the same hash value) is recorded, and a *forwards* comparison can be attempted for the purpose of finding the longest common substring in both the source and the target text.

  Comparison in the source and the target text continues at the next token position respectively from the ones defined above. Each token in the source text is compared to each token in the target text. If their hash values are equal, the size of the common substring (`size_best`) is recorded, and the next token is read. Comparison goes on for as long as the tokens' hash values in both the source and the target text match.

  In case of a mismatch, the size of the longest common substring, which should be greater than or equal to the `Min_Run_Size` value, and greater than the value of the LCS recorded so far (`size_best`), is returned, together with the target file `txt1`, and the index `i1` of the first token in the longest common substring sequence. The match is recorded by a call to the method `add_run()`, and comparison continues by reading the next eligible token in the source text, as described in the method `compare_one_text()`.

- If the hash value of a token in the source and the target sequence does not match, comparison in the target text continues at token position `Forward_Reference(i1)`, i.e. the next token position in the *chain* of tokens, at which a sequence of `Min_Run_Size` tokens with the same hash value starts.

The **method `add_run()`** is responsible for recording the matches found. For each match, a structure `run` is created, which contains a quality description, i.e. the size of the longest common substring, and two structures of type `chunk`: one for the source file, and one for the target file, which may be the same if the same file is under comparison. Each structure `chunk` holds information about the text (structure `text`) in which the match was found, as well as the first (structure `position`) and the last (structure `position`) token position of the match. The structure `run` is then inserted into an "arbitrary-in sorted-out" data type (`AISO`) according to its quality

description. Each of the structures of type `position` in both structures of type `chunk` of the structure `run` are stored into the linked list `tx_pos` of the corresponding file's structure `text`.

3.2.4 Deleting the forward reference table

After having recorded the matches in all input texts, the array `Forward_Reference` is of no use any more, and thus it is discarded. Allocated memory is subsequently freed.

3.3 Pass 2 and 3

Pass 2 and 3 are related to the issue of printing the matches recorded to the console. In this sense, these two phases are not considered to be an integral part of the *sim_text* algorithm, since they deal with an isolated matter, namely the presentation of the comparison results to the console according to a very concrete layout.

For this reason, a short documentation of each phase will be provided in the subsections to follow.

3.3.1 Pass 2

During this phase, the starting and ending line numbers of each chunk are recorded. For each file (structure `text`) that contains meaningful chunks, i.e. the linked list `tx_pos` is not null, the positions stored in this list are sorted by line number in ascending order. The positions' list, and the file are scanned in parallel, and the information of each position, depending on whether it is a starting or an ending position, is updated accordingly. If the array `nl_buff` exists, i.e. if, during pass 1, the amount of the available memory has been considered to be sufficient for recording the newline positions, accessing the file is abandoned, and the data stored in the array `nl_buff` are used to update the information of each position.

3.3.2 Pass 3

During this phase, the contents of the matches are printed to the console.
The structures of type `run` stored in the `AISO` data type are retrieved in descending order. For each structure `run`, which consists of two structures of type `chunk`, the streams of the files stored in each one of the structures `chunk` are opened, and positioned using the line numbers at the beginning of each `chunk`. Finally, the matches are displayed.

4 SIMILARITY TEXTER: A text-comparison web tool

This section focuses on the development of *similarity texter*, a text-comparison web tool whose implementation is based on the previously documented *sim_text* algorithm (see Section 3). It discusses the concept, as well as the various aspects related to the actual development of this tool, including the graphical user interface, the features, and the problems encountered during development, as well as the respective solutions found.

4.1 Concept

The original idea for the creation of this application should be credited to my supervisor, Debora Weber-Wulff. Having used Dick Grune's *SIM* extensively for tracing plagiarism in academic papers, she suggested the development of a web tool that would provide a more user-friendly environment, and improved visualization experience than the one in *SIM*, while using the same text comparison algorithm. The grounds of such a suggestion are justified by the following two considerations:

a) Web applications are gaining in popularity, mainly due to their supreme portability; users are spared from the arduous task of having to locally pre-install a program before running it. Furthermore, the development of faster processors, together with the continuously improved performance of modern web browser clients, makes it possible to execute complex tasks from within a web browser environment at speed rates comparable to the ones of a desktop application. Last but not least, web applications are more versatile in terms of graphical visualization; the introduction of CSS3, the latest standard for Cascading Style Sheets (CSS), combined with the abundance of free JavaScript (JS) plugins, enable the creation of an elaborate graphical user interface, which can be both more eye-catching, and user-friendly.

b) Several desktop applications, such as *JPlag*, *SIM*, and *WCopyfind*, that provide efficient results in terms of tracing lexical similarities in text input are available

for free download and use. However, this is not the case when it comes to online web applications. A research on the Web revealed the following, rather short, list of free web tools, which report string matches, either expressed as a percentage, or in a two-column layout, where matches are highlighted in color:

- **Copyscape**

  It compares URLs or plain text input.

  It outputs results side by side; matches are highlighted in light blue background color, and the total number of words per match is reported.

60 matching words were found:

Item 1 139 words, 43% matched		Item 2 135 words, 44% matched
The Project Gutenberg EBook of	« 6 words »	The Project Gutenberg EBook of
Ulysses, by James Joyce		The Adventures of Tom Sawyer, Complete by Mark Twain (Samuel Clemens)
This eBook is for the use of anyone anywhere at no cost and with almost no restrictions whatsoever. You may copy it, give it away or re-use it under the terms of the Project Gutenberg License included with this eBook or online at www.gutenberg.	« 46 words »	This eBook is for the use of anyone anywhere at no cost and with almost no restrictions whatsoever. You may copy it, give it away or re-use it under the terms of the Project Gutenberg License included with this eBook or online at www.gutenberg.

Figure 5. Output of the *Copyscape* web tool

- **Similarity Analyzer**

  It compares URLs.

  It outputs HTML code, and plain text similarities, expressed as percentages.

Compared pages:
First page: http://tvxs.gr/news/ellada/entos-ianoyarioy-i-rythmisi-gia-plastiko-xrima-kai-aforologito
Second page: http://tvxs.gr/news/ellada/i-xreokopia-xtypa-tin-porta-toy-dimoy-thessalonikis

	First page	Second page	Similarity
HTML fingerprint:	000081a81b01a111	000081a81b01a111	100.00%
HTML distribution value:	8e 00 00 00 04 00 00 00 a1 00 00 00 00 01 00 01 07 00 00 00 00 00 00 00 12 05 00 69 09 00 00 00 00 00 00 09 00 1d 00 1c 04 00 00 00 00 00 00 12	8c 00 00 00 04 00 00 00 a1 00 00 00 00 01 00 01 07 00 00 00 00 00 00 00 12 05 00 67 09 00 00 00 00 00 00 09 00 1d 00 1c 04 00 00 00 00 00 00 12	99.35%
Total HTML similarity:	-	-	**99.67%**
Standard text similarity:	-	-	56.84%
Smart text similarity:	-	-	39.67%
Total text similarity	-	-	**48.26%**

Figure 6. Output of the *Similarity Analyzer* web tool

- **String Similarity Test**

  It compares plain text input.

  It outputs string matches, expressed as a percentage.

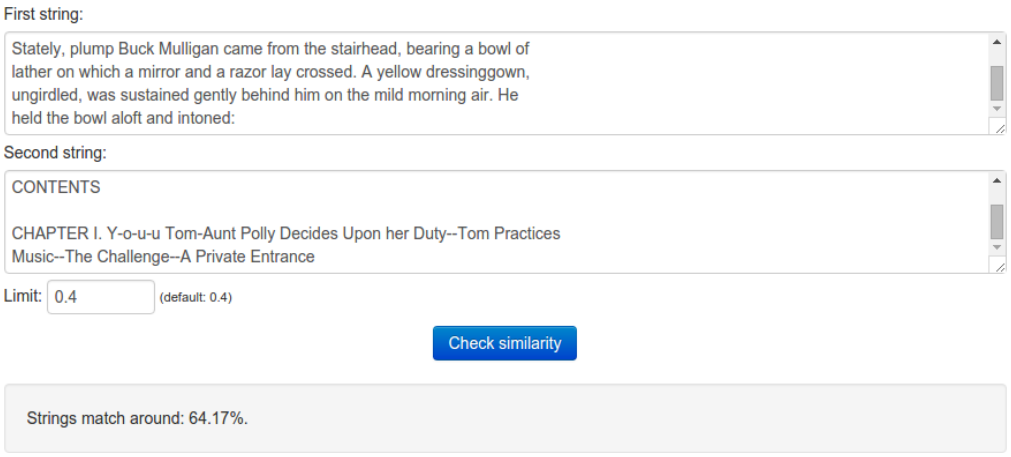

Figure 7. Output of the *String Similarity Test* web tool

- **Text Comparison**

  It compares HTML, and plain text input.

  It outputs results side by side; each match is highlighted in the same background color.

  Its implementation is based on the *sim_text* algorithm, developed by Dick Grune.

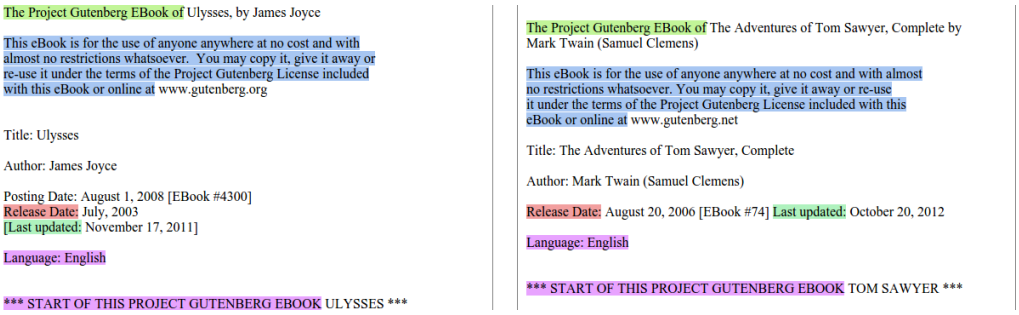

Figure 8. Output of the *Text Comparison* web tool

All four web tools are capable of measuring lexical similarities; however, they can be regarded as primitive in terms of user experience, since their output is static, and they do not provide any kind of user interaction.

4.2 Defining the functional requirements

An important step before diving into the actual development of a program is to discover all kind of interactions between a system, and its actors/users. Use case diagrams help the developer identify possible scenarios, and subsequently derive the functional requirements, i.e. the features, that a system should possess (Sommerville, 2011).

Figure 9 illustrates a number of use cases that constitute the major functions initiated by the user. As can be concluded, the user should be able to perform the following interactions with the system:

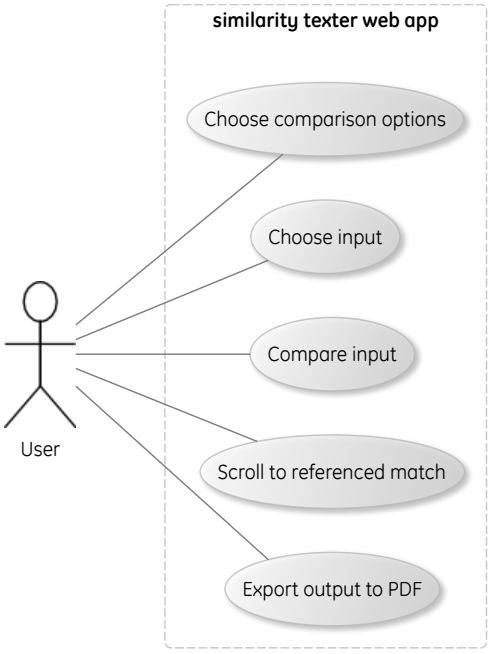

Figure 9. Use case diagram of *similarity texter*

Use case	Functional requirement
1. Choose comparison options	Users should be provided with a set of options that fine-tune the way in which comparison is performed. They should be able to select the ones of their choice, which should be taken into account during comparison. ⇒ **Tokenize input based on comparison options**
2. Choose input	Users should be able to select among different types of input (e.g. DOCX, ODT, and TXT files, as well as HTML/plain text) the ones, namely two in total, which should be included in the comparison. Support should be provided for any character set in the Unicode standard, encoded in UTF-8. ⇒ **Read input**

Use case	Functional requirement
3. Compare input	Users should be able to conduct a comparison between input of the same or different type, according to the selected comparison options. The implementation should be based on the *sim_text* algorithm, developed by Dick Grune. ⇒ **Compare input**
4. Scroll to referenced match	Users should be provided with a convenient way of inspecting the results returned by the comparison. In this sense, automatic alignment between the source and the target highlighted match should be triggered, when clicking on either the source or the target highlighted match. ⇒ **Auto-scroll to target highlighted match**
5. Export output to PDF	Users should be provided with the option of generating a PDF report from the contents of the comparison output. They should also be able to include a comment of their choice in the report. ⇒ **Generate PDF report**

4.3 Designing the graphical user interface

The graphical user interface of a software product plays a central role as regards user experience (UX). The proposed solution, as shown in Figure 10, attempts to organize the layout in such a manner so that it provides maximum user experience, both physical as well as emotional.

The design of the layout takes into account the *Gestalt principles of perception* (Lowdermilk, 2013). In this regard, elements with close relationships are placed together (*principle of proximity*), whereas elements denoting the same functionality/capacity share the same visual characteristics (*principle of similarity*).

The layout consists of the following distinct sections:

1. the static *navigation bar*, which contains navigation links and toggle buttons,
2. the togglable *settings sidebar*, which contains the settings of the web application,
3. the togglable *input panel*, which provides two input panes for uploading different types of input, and
4. the *output panel*, which provides two output panes for displaying the results of the comparison side by side.

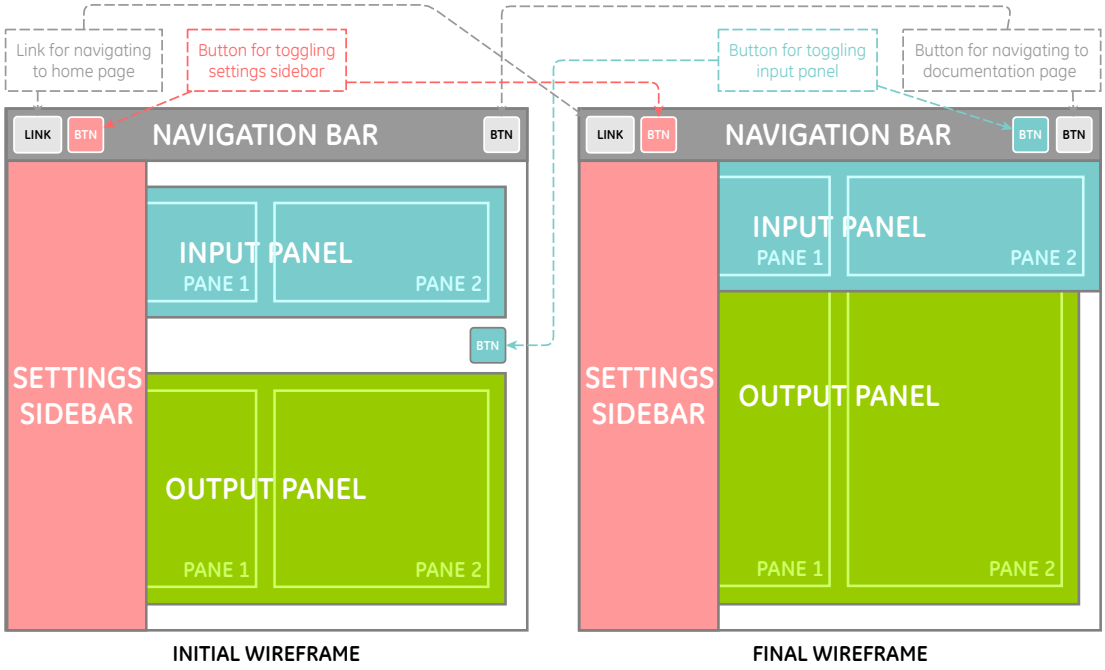

Figure 10. Initial and final wireframe of the GUI

Figure 10 illustrates the two different versions of wireframes conceived for the realization of the GUI. As can be seen, their main difference lies on the positioning of the input panel. In the initial wireframe, the input panel has a relative position above the output panel; users can toggle its visibility by pressing the corresponding button located at the top right corner of the output panel. On the contrary, the input panel, in the final version, has an absolute position below the navigation bar. This change is justified due to the following reasons:

a) **Usability:** The final version provides a solution which involves a smaller number

of clicks from the part of the user: in the initial version, users have to press the toggle button for hiding the input panel, and thus maximizing the output panel. In the final version, the input panel gets automatically hidden, once the comparison output is displayed.

b) **Performance:** When toggling the input panel in the initial version, the entire DOM gets repainted. This operation is expensive, and the time required for its completion depends on the number of node elements contained in the DOM. Consequently, repainting a large number of node elements, which may be a common case in *similarity texter*, will cause the web browser to freeze. The final version of the GUI redresses this issue through the absolute positioning of the input panel.

The web design of the final layout is also responsive, which means that it supports the presentation of the web application on different devices (e.g. desktops, tablets, and phones), and screen sizes respectively.

For the development of the GUI's styling, the following external UI frameworks/plugins have been used:

- the *Bootstrap* UI framework, and
- the *Bootstrap FileStyle* plugin, which customizes the `<input type="file">` control element for *Bootstrap*.

4.4 Development

Similarity texter is written in JavaScript, i.e. the native language for client-side web development that is supported by all web browser clients. The reason for choosing this programming language for the development of the web application's backend is mainly based on the ease that this option provides in terms of deployment. More specifically:

- To deploy the web application on a remote web server for *online* use, the folder `dist` of the application's built version should be uploaded to the server, and a URL should be assigned to it, pointing to the main HTML page of the application (i.e.

`app.html`). To run the application, users are required to navigate to the specified URL from within their web browser.

- To deploy the web application on a local system for *offline* use, the folder `dist` of the application's built version must be present on the local file system. To run the application, users just need to click on the main HTML page of the application (i.e. `app.html`).

For the development of the different features of the web application, external JS libraries have been used. These include:

- the *jQuery* library, which simplifies the client-side scripting of HTML, and
- a number of other JS libraries, each one of which will be explicitly referenced in the subsection that documents the feature involved.

The source code is structured in modules, and compiled into a single JS file with the use of *browserify*; *browserify* is an external JS tool, based on *Node.js*, for bundling up JS dependencies (for recompiling the code, see file `README` in folder `similarity-texter`).

The actual implementation of each functional requirement, as defined in Subsection 4.2, is discussed in the subsections to follow.

4.4.1 Reading the input

An important feature of *similiarity texter* is that it supports the comparison of different types of input: DOCX, ODT, TXT file formats, as well as HTML and plain text. As shown in Figure 10, the graphical user interface provides two tabbed input panes, where users are able to select the type of input they wish to compare to each other. The use of tabs makes it possible to choose different types of input in the source and the target pane.

To handle the particularities of each type of input, different input readers have been implemented. The module `fileInputReader.js` (see Source file 14) is responsible for extracting the text contents of the aforementioned file formats, and the module `textInputReader.js` (see Source file 15) handles the text extraction of the HTML input.

The reading of both types of input (i.e. file and text) is performed asynchronously, so that to prevent the web browser from freezing.

It should be noted that TXT files must be encoded in UTF-8 in order for their text contents to be read properly.

File input reading

File input reading involves the successful execution of the following number of steps: accessing the local file, reading its data, and extracting the text contents depending on the selected file format. The fifth revision of the HTML standard (HTML5) supports interaction with local files via the File API (W3C, 2015). More specifically, the `File` interface provides methods for accessing local files, whereas the `FileReader` interface handles the reading of their data.

Accessing the local file

The standard way to access a local file is through the use of an `<input type="file">` control element. Each input pane of the GUI, under the tab "FILE", contains a control element of this type, whose purpose is two-fold: first, it allows users to select, from their local directory, the file they wish to upload for comparison, and second, it grants permission to the web application to access the users' local file system. For obvious security reasons, web pages are not allowed to access to the users' local directories, unless an explicit permission is granted.

When the user selects a file through this control element, an array-like collection of `File` objects is returned, which can be accessed via the property `files` of the control element. Each `File` object provides read-only information about each file being selected, such as the name, the file size, the MIME type, as well as a reference to the file itself. It should be noted that, in our case, this collection contains only one `File` object, since the selection of multiple files is not supported.

Reading the file's data

Once the file is obtained, its data can subsequently be read. For this purpose, a `FileReader` object is instantiated, which provides methods for the asynchronous

reading of different types of file data. The method `readAsText()` returns the file contents as plain text, and therefore has been used for the reading of TXT files. The method `readAsArrayBuffer()` returns the file contents as an `ArrayBuffer` object (i.e. a raw binary data buffer), and thus has been employed in the case of DOCX and ODT file formats.

The `FileReader` interface also provides a set of event handler attributes, which monitor the progress of the reading operation. The `onerror` event has been used for tracing potential errors during reading, the `onloadstart` event for tracking when the actual reading starts, and the `onload` event for tracking the successful completion of the reading operation. When the `onload` event is fired, the file's data are returned, which can be accessed via the property `result` of the `onload` event.

Extracting the text contents of DOCX, ODT and TXT file formats

As regards TXT file formats, the extraction of their text contents can be considered as completed at this point, since the property `result` returns the file contents as a string. However, when reading TXT files created under OS X systems, line breaks are not processed properly; they are actually ignored. This is based on the fact that each operating system produces different line endings during the creation and/or editing of a TXT file. For instance, Unix systems use the line feed symbol `\n` for newlines, Windows systems the carriage return, together with the line feed, symbol `\r\n`, and OS X systems just the carriage return symbol `\r`. To redress this issue, all carriage return symbols in the string `result`, which are not followed by a line feed symbol, are replaced by a line feed symbol.

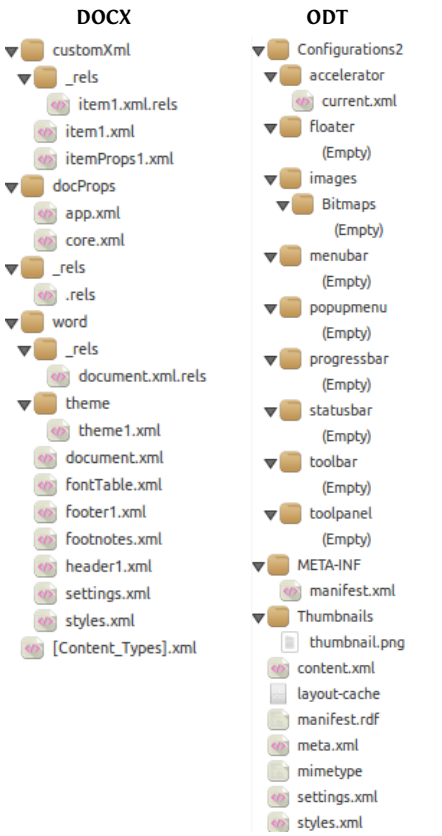

Figure 11. Typical package structure of DOCX and ODT files

As regards DOCX and ODT files, the property

`result` returns an `ArrayBuffer` object, namely a fixed length buffer of bytes, which requires further processing. This task is carried out by the external JS library *JSZip*, which supports the creation, reading, and editing of ZIP files.

The selection of this JS library is justified by the fact that DOCX and ODT files are basically ZIP packages, which consist of a number of parts, namely of XML files, and conform to a concrete structure, as specified by the ECMA-376 standard for WordprocessingML (ECMA Internaltional, 2012), and the OASIS standard for OpenDocument (OASIS Open, 2011) respectively. Figure 11 illustrates the typical package structure of an unzipped DOCX and ODT file format.

Each XML-based file holds different parts of information about the document, which are interrelated to each other, such as the settings, the styling, the media files (e.g. images and videos), and the body of the root document, among others. Given that we are only interested in reading the document, together with its footnotes/endnotes, the XML files which are relevant in our case are listed in the following tables:

DOCX	**Description**	**Basic structure**

`word/document.xml`	Contains the body of the main document.	`<w:document>`
		`  <w:body>`
		`  <w:p />`
		`  </w:body>`
		`</w:document>`

`word/endnotes.xml`	Contains the endnotes of the main document.	`<w:endnotes>`
		`  <w:endnote />`
		`</w:endnotes>`

`word/footnotes.xml`	Contains the footnotes of the main document.	`<w:footnotes>`
		`  <w:footnote />`
		`</w:footnotes>`

ODT	**Description**	**Basic structure**

`content.xml`	Contains the body of the main document, as well as its endnotes and footnotes (if any).	`<office:document-content>`
		`  <office:scripts />`
		`  <office:font-face-decls />`
		`  <office:automatic-styles />`
		`  <office:body>`
		`  <office:text>`
		`  <text:sequence-decls />`
		`  <text:p />`
		`  </office:text>`
		`  </office:body>`
		`</office:document-content>`

As mentioned above, the *JSZip* library is used for extracting the package contents of these two zipped file formats. Depending on the selected type of file, and on the enabled input reading options (i.e. whether the footnotes/endnotes should be ignored or not), the stream of bytes of the corresponding XML-based file(s), as defined above, is read, and converted into a string. The resulting string is then parsed into a valid XML document using the *jQuery* method `parseXML()`, and the operation of extracting the text contents begins.

Text extraction takes into account the differences in the markup, as well as in the XML document's structure of each file format. Therefore, different selectors, as described in the following tables, are used for selecting those nodes, which are of interest, for each type of file, and XML document involved.

DOCX	**Top selectors**	**Child selectors (apply to all top selectors)**	
`word/document.xml`	`w:body`	`w:p`	Specifies a paragraph content.
`word/footnotes.xml`	`w:footnotes`	`w:t`	Specifies a text content.
`word/endnotes.xml`	`w:endnotes`	`w:br`	Specifies a break.

ODT	**Top selectors**	**Child selectors (apply to all top selectors)**	
`content.xml`	`office:body`	`text:p`	Specifies a paragraph content.
	`text:note-body`	`#text`	Specifies a text content.
		`text:line-break`	Specifies a break.

The nodes of each XML document are first filtered out according to the top selector, as specified in the tables above. The resulting array of nodes is traversed recursively, as follows:

For each node in the array of nodes:

1. Check whether it contains child nodes.
   If it does, step into the first child node, and check its type:
   1.1. If it is a text element (selector `w:t` or `#text`), store its text content, and step into the next sibling of this child node. Repeat the process from step 1.
   1.2. If it is soft line break element (selector `w:br` or `text:line-break`), store a

line feed symbol `\n`, and step into the next sibling of this child node. Repeat the process from step 1.

1.3. Else, starting from this child node, repeat the process from step 1.

The selectors `w:p` and `text:p` assure the insertion of a line break, once the text content of all child nodes, contained in these two node elements, has been extracted.

Text input reading

The graphical user interface contains a `<textarea>` control element in each input pane, under the tab "TEXT", where users can provide HTML or plain text input. Each time a user types or pastes text in these control elements, an event listener is triggered, which returns the input provided as a string.

The problem that the implementation has to solve arises from the dual character of the input: namely, how to determine whether the input provided is plain text, which needs no further treatment, or HTML text, which requires to be further processed.

The first approach, which has been followed, uses of the *jQuery* method `parseHTML()` to parse the input string into an array of DOM nodes, and then checks whether the resulting array contains only nodes of type 3 (i.e. text nodes). In such a case, it can be assumed that the provided input is plain text, and no further processing is required. However, this approach does not always deliver the correct results, thus leading to a severe truncation of the input. The reason of this truncation lies on the fact that this *jQuery* method does not validate the HTML string. Consequently, if a plain text input contains, for instance, a word surrounded by angle brackets `< >`, this word will be interpreted as a node of type 1 (i.e. element node) by the method, and our original assumption collapses.

To overcome this issue, a different approach has been followed. The main idea is to treat both types of input (i.e. HTML and plain text) as HTML. In this respect, an element node `<div>` is created, and the input string is assigned to its content (property `innerHTML`). The text contents of this element node are then parsed, following the same logic as the one described for the reading of files: all nodes, contained in the element node `<div>`, except for those which are web-specific (namely `iframe`,

`noscript`, `script`, and `style`), are traversed recursively, and their text content (node of type 3) is extracted and stored as a string. However, this approach "suffers" from exactly the same problem as the first one, but to a smaller extend. For instance, if the user provides some plain text input, which contains words surrounded by angle brackets < >, these words will be interpreted as element nodes, and thus they will be excluded from the resulting string.

The only safe solution to the dual-input problem is to let the user decide over the type of input. In this regard, a checkbox control element ("HTML") has been added to each text input pane. If it is checked, the provided input is treated as HTML, and its text contents are extracted following the method described in the second approach. Otherwise, the provided input is treated as plain text, and thus no further processing is required.

4.4.2 Comparing the input

Text comparison constitutes the core feature of this web application. According to the functional requirements in Subsection 4.2, its implementation should be based on the *sim_text* algorithm, as analyzed, and documented in Section 3. In this respect, it should take into account the specificities of both the target programming language (i.e. JavaScript) and the web application in question, and adapt the algorithm accordingly, by including, excluding, and/or adjusting the steps involved as necessary.

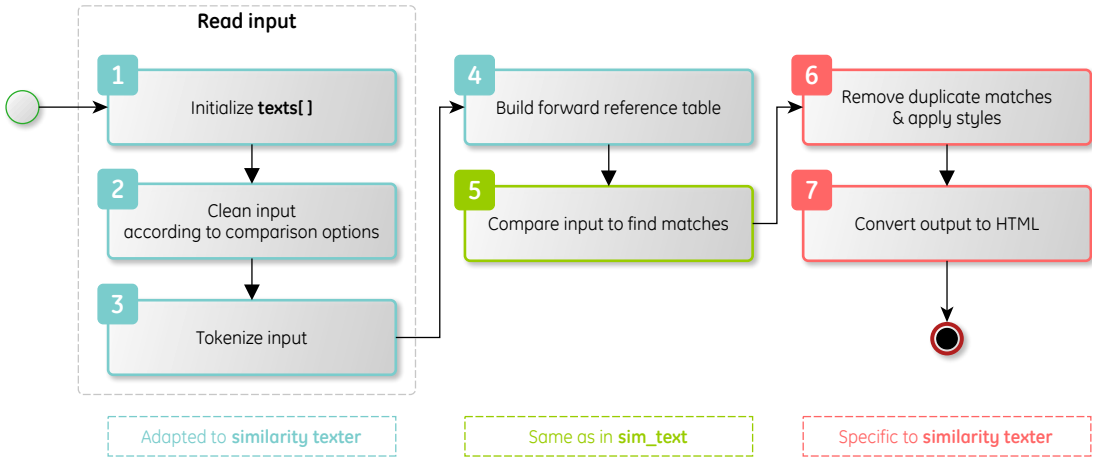

Figure 12. Activity diagram of the module `SimTexter.js`

The module `SimTexter.js` (see Source file 18) attempts to port the *sim_text* algorithm, written by Dick Grune in C, to the JavaScript programming language. Figure 12 provides an overview of the steps implemented in JS, while highlighting in different colors the degree of similarity between the C and JS implementation of each step.

The following table summarizes the similarities/differences of the two implementations, taking as a starting point for the comparison the phases, together with their steps, of the *sim_text* algorithm, as described in Section 3.

Phase	Step
Pass 1	**1. Reading the input files** This step is implemented in steps 1, 2, and 3, as shown in Figure 12. To accommodate the functional requirements of the web application in question, the implementation has been adapted as follows: • Algorithm 1, which calculates the hash code of a word, has been excluded from the implementation, since it has been implemented for, and tested against a restricted character set (i.e. extended ASCII table). On the contrary, *similarity texter* should provide support for all character sets encoded in UTF-8. • As opposed to *sim_text*, parsing and tokenization of the input in *similarity texter* should take into account the comparison options set by the user.
	2. Building the forward reference table This step is implemented in step 4, as shown in Figure 12. The implementation follows the same algorithm as in *sim_text*, except for Algorithm 2 and 3 (hash functions), which are not implemented for the same reasons described above.
	3. Comparing the input files This step is implemented in step 5, as shown in Figure 12. Implementation follows the same algorithm as in *sim_text*.

Phase	Step
	4. Deleting the forward reference table This step is excluded from the implementation, since it is specific to the C programming language.
Pass 2 and 3	These two phases, including their steps, are excluded from the implementation, since they deal with a *sim_text*-specific issue, namely the printing of the matches to the console.

A description of the actual JS implementation of the steps, illustrated in Figure 12, follows below.

1. Initializing the array texts

As mentioned in Subsection 4.4.1, the input reading operation returns the text contents of each input (i.e. source and target) as a single string. For each input string, an object `Text` is created, which holds the following information:

- the mode of input (property `inputMode`), which defines whether the input is a file or a text,
- the total number of characters of the input string (property `nrOfCharacters`),
- the total number of words of the input string (property `nrOfWords`),
- the name of the file (property `fileName`), if the input is a file; otherwise, the value "HTML text input" or "Plain text input", depending on whether the input provided is HTML or plain text, is assigned to this property,
- the index (inclusive) of the object `Token` in the array `tokens`, at which the input string starts (property `tkBeginPos`), and
- the index (non-inclusive) of the object `Token` in the array `tokens`, at which the input string ends (property `tkEndPos`).

Each object `Text` is then added to the array `texts`.

2. Cleaning the input

During this step, the comparison options enabled by the user are parsed; depending on their values, different regular expressions are built to match the input

string against the rules defined by these options. It should be noted that, for the recognition of the Unicode characters which stand for numbers and punctuation symbols, the external JS library *XRegExp* has been used; besides the augmented and extensible JS syntax for regular expressions that it provides, it includes add-on packages (see http://xregexp.com/plugins/), which list Unicode characters under named categories (e.g. letter, number, punctuation, etc.).

The rules defined by the comparison options are described below:

- **Rule "ignore letter case"**

  When the variable `ignoreLetterCase` is enabled, the input string is converted to lowercase; the length of the input string remains unchanged.

- **Rules "ignore numbers"** (*XRegExp* category alias: `Number`) **and "ignore punctuation"** (*XRegExp* category alias: `Punctuation`)

  When the variables `ignoreNumbers` and/or `ignorePunctuation` are enabled, all numbers and/or punctuation symbols respectively, contained in the input string, are replaced by a space. Were the character deleted from the input, this would alter the original positions of all subsequent characters in the input string, which is not desirable before tokenization.

- **Rule "replace umlaut & ligatures"**

  When the variable `replaceUmlaut` is enabled, the following umlauted characters and ligatures are replaced by their equivalent expanded versions. This rule alters the length of the input string, and thus it is applied during tokenization.

Character	Replaced by
ä, æ	ae
ö, œ	oe
ü	ue
ß	ss

Each input string gets "cleaned" according to those rules which do not alter the original length of the string.

3. Tokenizing the input

During this step, each "cleaned" input string is tokenized into words. A *word* is an arbitrary sequence of characters/symbols, separated by one or more white space characters (i.e. space, tab, line break). For each recognized word in the "cleaned" input string, an object `Token` is created, which records the following information:

- the word's starting (inclusive) and ending (non-inclusive) character position, absolute to the input string (properties `tkBeginPos` and `tkEndPos` respectively). These two values simplify the conversion of the output (i.e. text and matches) to an array of HTML nodes;
- the "cleaned" word (property `text`), i.e. the resulting text of the word after being matched against the pattern defined by the rule "replace umlaut & ligatures".

Each object `Token` is then added to the array `tokens`.
Upon completion of the tokenization process, the property `tkEndPos` of each object `Text` in the array `texts` gets updated.

4. Building the forward reference table

Except for the hash functions which are not implemented for the reasons stated above, the creation of the forward reference table follows the same algorithm as in *sim_text*. The hash table `last_index` is implemented as an associative array in JS.

5. Comparing the input

This step is implemented using the same algorithm as in *sim_text*. Comparison returns an array of matches, where each match is an array of two `MatchSegment` objects stored in pairs; at index `0` of this array the source input's `MatchSegment` is stored, and at index `1` the corresponding `MatchSegment` of the target input. Each object `MatchSegment` stores the following information:

- the index of the object `Text` in the arrays `texts`, at which the match has been found (property `txtIdx`),
- the index of the object `Token` in the arrays `tokens`, at which the match starts (property `tkBeginPos`),

- the length of the match (property `matchLength`), and
- the style class to be applied to the object `MatchSegment` during step 6.

6. Removing duplicate matches and applying styles

During this step, any duplicate matches found in the array of matches, returned by the comparison operation, are discarded, whereas overlapping matches get shrunk in length. This is mainly due to the restrictive HTML markup syntax, which does not allow the overlapping of node elements (e.g. `<a>match one overlaps with <a>match</a> two</a>`). In addition, the style class of each match is set accordingly.

To deal with these issues, the array of matches is sorted by the target `MatchSegment` object (i.e. index `1`) as follows: its property `tkBeginPos` is sorted in ascending order, and its property `matchLength` in descending order. An example of a sorted array of matches is given below:

```
matches[0][1]: tkBeginPos:  1, matchLength:  4
matches[1][1]: tkBeginPos:  7, matchLength: 10
matches[2][1]: tkBeginPos:  7, matchLength:  7
matches[3][1]: tkBeginPos:  7, matchLength:  4
matches[4][1]: tkBeginPos: 15, matchLength:  4
```

The reason for opting for this kind of sorting is based on the following observation: as shown in the above example, the first element in the order of target `MatchSegment` objects with the same `tkBeginPos` value has the longest length; consequently, all other elements with the same `tkBeginPos` value can be ignored, since they are either duplicate or overlapping matches.

For storing the unique matches, an array is created. The first element of the original array of matches is added to it; this serves as a starting point for the comparison that will follow. For each element in the original array of matches, the target `MatchSegment` object is compared to the last target `MatchSegment` object stored in the array of unique matches.

- If they are duplicates, i.e. their properties `tkBeginPos` have equal values, the current match is not added to the array of unique matches, and the next match

in the original array of matches is read.

- If they are not duplicates, a second check is carried out in order to determine whether the current match is overlapping with the last stored unique match.
  If they do not overlap, i.e. the token's end position (i.e. `tkBeginPos + matchLength`) of the last target `MatchSegment` object stored in the array of unique matches is less than the value `tkBeginPos` of the current match's target `MatchSegment` object, the current match, i.e. the current array of source and target `MatchSegment` objects, is added to the array of unique matches, and their style classes are set accordingly.
  If they are overlapping, i.e. the token's end position of the last target `MatchSegment` object stored in the array of unique matches is less than the token's end position of the current match's target `MatchSegment` object, the last stored unique match gets updated as follows:
  - the property `matchLength` of its target `MatchSegment` object gets shrunk, and
  - the style class "overlapping" is added to the property `styleClass` of both the source and the target `MatchSegment` objects.

  In addition, the current match is added to the array of unique matches, and its style classes are set accordingly.

7. Converting the output to HTML

To visualize the results of the comparison in HTML, the entire text of each input, together with the matches found, is converted into an array of HTML nodes in one pass, as follows:

For each input string (source or target), an array is created for storing its node elements. The array of unique matches, returned during step 6, is then sorted by source/target `MatchSegment` object respectively, using the same sorting method as the one described above.

The input string is read by character position (`chIdx`). Starting at position 0 (first character of the string), each source/target `MatchSegment` object in the array of unique matches is read, and the positions of its first (`mTxtBeginPos`) and last (`mTxtEndPos`) character in the input string are calculated. The computation of

these two values is simple, since they can be derived from the starting and ending token positions stored in the object `MatchSegment`.

The next step involves the creation, exactly in this order, of two distinct node elements, namely:

a) a text node element, which will hold the text content of the segment that precedes the current source/target match segment, and
b) an element node `<a>` (i.e. link), which will hold the text context of the current source/target match segment.

To extract the text content of the first node element, the input string is sliced at starting position `chIdx` and ending position `mTxtBeginPos`. The resulting substring is appended to the newly created text node element, and the latter is added to the array of node elements.

To extract the text content of the second node element, the input string is sliced at starting position `mTxtBeginPos` and ending position `mTxtEndPos`. The resulting substring is appended to the newly created element node `<a>`, and the following attributes are added to its markup:

- the attribute `id`, whose value equals the index of the match segment's starting token,
- the attribute `href`, which points to the attribute `id` of the target `MatchSegment` object, and
- the attribute `class`, which defines the styling of the node element (i.e highlighting color).

It should be noted that the attributes `id` and `href` are necessary for the implementation of the auto-scrolling functionality (see Subsection 4.4.3).

The node element is then added to the array of node elements, and the current character counter `chIdx` gets updated with the value of the variable `mTxtEndPos`.

The steps described above are repeated for each source or target `MatchSegment` object stored in the array of unique matches, and for as long as the last character of the input string has not been reached.

4.4.3 Auto-scrolling to the target highlighted match

To enhance user experience as regards the inspection of the comparison's results, the web application provides an integrated functionality which performs automatic alignment between the source and the target highlighted match as follows: clicking on a highlighted match on either output panes triggers auto-scrolling to the target match in the opposite output pane. Both matches are aligned at the same level, thus providing a better overview of their content.

The implementation of this functional requirement is divided in the following distinct parts.

HTML structure of the output panes

As shown in Figure 13, the HTML document follows a specific structure, which is mandatory for the calculation of the target match's new scroll position.

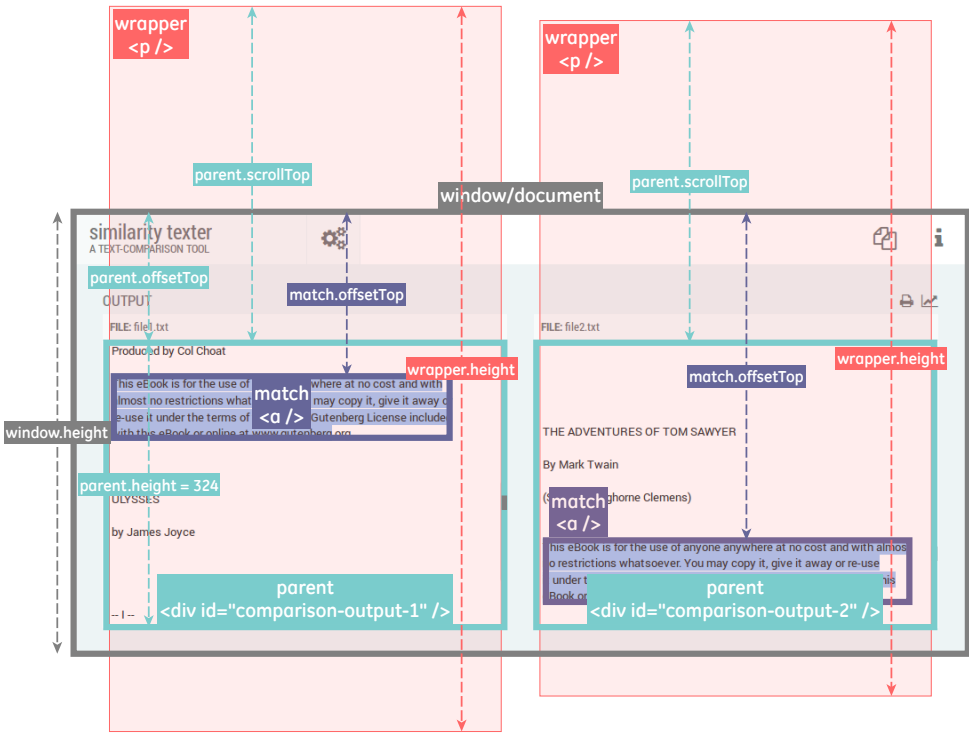

Figure 13. HTML structure for the auto-scrolling feature

The `document` object is the root node element, under which all other nodes lie. In our case, its height equals the height of the window, since its element node `<div id="#content-wrapper">` is defined as non-scrollable, i.e. it does not have a scroll bar (CSS style `overflow: hidden`).

Each element node `<div>` (hereinafter referred to as "node *parent*"), in the source and the target output pane respectively, is a child node of the element node `<div id="#content-wrapper">`; its height remains unaltered irrespective of the total height of its child nodes. In addition, it is defined as a scrollable area on the *y* axis (CSS style `overflow-y: scroll`).

Each element node `<p>` (hereinafter referred to as "node *wrapper*") is the direct child node of each element node `<div>`, and the parent of all underlying nodes, which contain the text and the highlighted matches (i.e. text nodes and element nodes `<a>` respectively). It does not have a scroll bar, and its height depends on the total height of its child nodes. This element node is required for the implementation of the auto-scrolling feature, since it provides information on the actual total height of the underlying nodes.

HTML markup of the element nodes <a>

As mentioned in Subsection 4.4.2, each match returned from the comparison operation is converted to an HTML element node `<a>`. An example of the resulting HTML markup for a source and a target match is given below:

```
<a id="1-16"   href="#2-2084" class="hl-1">source match text</a>
<a id="2-2084" href="#1-16"   class="hl-1">target match text</a>
```

According to the HTML markup specification, the attribute `href` specifies the link's destination. Consequently, binding the source match's attribute `id` to the target match's attribute `href` enables the navigation to the target match when clicking on a source match, and vice versa.

Calculation of the target match's new scroll position

Auto-scrolling to a target match entails that a new position is assigned to the scroll bar of the target node *parent*. For the calculation of this value, the *jQuery* library

provides the following two helpful methods:

- The method `offset().top` returns the current top position (i.e. *y* coordinate), relative to the `document`, of the selected element, and
- the method `scrollTop()` returns the current vertical position of the scroll bar for the selected element, i.e. the number of pixels hidden from view above the scrollable area.

To calculate the new position of the scroll bar for the target node *parent*, the top position of the source match (selected element node `<a>`) is subtracted from the top position of the target match (referenced element node `<a>`). The resulting difference, which may be a positive or a negative number depending on the current top position of the target match, is then subtracted from the current position of the scroll bar of the target node *parent*.

However, the solution suggested above does not take into account the case where a target match is located at the very beginning or the very end of the target node *wrapper*. In such a case, the two matches cannot be aligned at the same level, since the calculated new scroll position for the target node *parent* exceeds the top or the bottom position respectively of this node. To redress this issue, top or bottom padding is added to the target node *wrapper*, as necessary.

Top padding is added if the calculated new scroll position is less than 0. In such a case, the value of the top padding to be added equals the absolute value of the new scroll position. After the addition of the top padding, the value of the scroll position is set to 0, i.e. top scroll position.

Bottom padding is added in the case where the calculated new scroll position exceeds the bottom position of the scrollable area (namely the bottom of the vertical scroll bar). This position is given by subtracting the height of the node *parent* from the height of the node *wrapper*. To compute the bottom padding, the height of the node *wrapper* is subtracted from the actual position of the target match (referenced element node ‹a›), which is relative to the parent node, namely to the node *wrapper*. This position is given by adding the calculated new scroll position to the height of the node *parent*.

4.4.4 Generating the PDF report

According to the list of functional requirements (see Subsection 4.2), users should be able to generate a PDF report that includes the contents of the comparison output. Furthermore, it should be possible for them to add comment(s), one for each input under comparison, to the report.

As shown in Figure 14, the structure of the PDF report should conform to the following formatted layout, comprising of:

- a **header**, which may hold a title, e.g. the name of the web application,
- a **footer**, which holds the number of the current page, and
- a **body**, which contains the following parts:

  Parts to be shown only on the first page

  – a section reserved for printing out the comment(s) inserted by the user, and

  – a section reserved for printing out statistical information on the input under comparison.

  Parts to be shown on all other pages, except for the first one

  – a section reserved for printing out the comparison output; the contents of the source and the target text, together with their highlighted matches, are printed side by side.

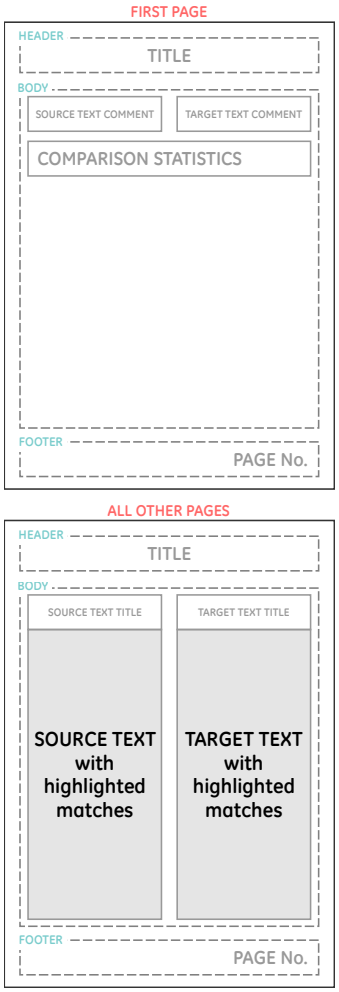

Figure 14. Layout of the PDF report

For the implementation of this feature, different approaches have been examined; these include:

- the use of an external JS library for PDF generation, and
- the use of CSS, together with the system's PDF printer.

PDF generation using an external JS library

The following libraries are the most complete open source JS libraries for PDF generation, which are available for free download, and use: a) *jsPDF*, b) *PDFKit*, and c) *pdfmake*. All three of them are able to produce elaborate PDF documents, given that they provide support for an extensive list of features, such as the selection of font faces, the application of different text styling, the inclusion of images, the creation of various shapes, and the insertion of annotations, to name a few.

As opposed to *jsPDF* and *PDFKit*, *pdfmake* comes with a more sophisticated layout engine; it supports the straightforward creation of paragraphs, tables, columns, among others, since text typesetting is automatically carried out by the library depending on the options set (e.g. page size, margins' size, column gap, etc.).

To create a two-column layout for printing out the contents of the comparison output, as shown in Figure 14, the developer just needs to populate a JS object that follows a concrete structure, as specified by the library's API, with the contents returned by the comparison. The positioning of each text segment on the page is done seamlessly by the library; the precise *x* and *y* coordinates of each segment, as well as the number of pages to be generated are automatically calculated in the background, and depend on the size and the margins of the page, the size of the column gap, and the type and the size of the font face specified. Furthermore, the library supports the application of different text styling (e.g. colored highlighting) to individual segments.

In terms of running time, *pdfmake* can produce efficient results for PDF documents with a relatively small number of pages. As stipulated in the issue no. 280 (see http://github.com/bpampuch/pdfmake/issues/280), the algorithm takes three seconds to generate a 20-page PDF document with a simple layout, but needs approximately two minutes for the double number of pages. This leads to the assumption that either the algorithm contains a bug, which has to be fixed, or its time complexity increases exponentially depending on the size of the PDF document.

Taking into account that *similarity texter* supports the comparison of text input that may amount to several hundreds of pages (with a suggested maximum of 500 pages per text input), the aforementioned solution cannot be regarded as an efficient one, and therefore it has been abandoned.

PDF generation using CSS and the system's PDF printer

Cascading Style Sheets (CSS) is a simple design language, which determines the way in which a web page is presented; in other words, it defines its look-and-feel. One of the most interesting features of CSS is that they allow for a web page to be presented differently on different types of media: on the screen, on paper, on braille or television devices, etc.

The approach of generating a PDF report by taking advantage of the this CSS feature, and the system's PDF printing capability is based on the following considerations:

1. A built-in PDF printer is installed by default in Linux, and OS X operating systems. As regards Microsoft, this feature has finally been included in Windows 10.
2. The layout of *similarity texter*, as presented on the screen (see Figure 10), resembles to a great extend to that of the PDF report (see Figure 14); namely they both follow a two-column structure. In this regard, customizing the styling of the printable version entails minimal style adjustments.

The presentation of the web page on a printable medium is customized by the following two CSS rules, which are both included in the file `media_print.less`:

- the `@page` rule, which specifies the page layout, and the pagination of the printed document, and
- the `@media print` rule, which specifies the new style rules for those HTML elements that need to be modified when the web page is directed to the printer. Unnecessary sections are hidden from the output, i.e. they are defined as invisible, whereas essential sections, such as the two output panes containing the comparison results, are modified, so that their styling matches that of the PDF report.

For printing out the statistical data and the user's comments, two sections are reserved in the DOM. Both of them are hidden on the screen, and become visible only in the printable version of the web page. The first section holds the statistical data, and gets updated upon completion of the comparison. The second section, which is related to the user's comments, gets updated once the user has provided some input in the corresponding text boxes of the "PRINT OUTPUT" pop-up form, and clicks on

the "PRINT" button.

By pressing the "PRINT" button, a call is being made to the system's print command. The default printing dialog appears, and the user is provided with a set of options, which can further adjust the current printable version of the document. For the proper creation of the PDF report, the following options **must** be enabled; the exact names of the options may vary depending on the operating system. These include:

- the option "Print to File", for directing the document to the PDF printer, instead of the default printer, for printing,
- the options "Color printing" and "Background colors", for displaying the colored highlighted matches, and
- the option "Headers and footers", for including a title and a page number on each PDF page.

In terms of running time, this approach has proven to be very efficient, and therefore has been used for the implementation of this functional requirement. More specifically, for the generation of a 500-page PDF document, Mozilla Firefox needs approximately 45 seconds on an Intel Core i7 (2nd generation) system, whereas Google Chrome needs twice as long, which can still be regarded as an acceptable time lapse for such a large amount of output.

Finally, it should be noted that the pagination rule `page-break-after` is not properly processed by Google Chrome, thus resulting in the generation of a PDF report that does not contain an explicit page break between the first and the second page. This issue has been reported by many users as a common bug (see http://productforums.google.com/forum/#!topic/chrome/qQ0wzHfgves).

The same issue is also recorded in Internet Explorer under Windows 7. In addition, the web browser fails to interpret correctly the CSS style `white-space: pre-line`, thus resulting in the wrapping of all paragraphs in the section "Comparison output". It should be noted that for the generation of the PDF report under this operating system, an external free PDF Writer has been used, namely *PDF24 Creator* (v.7.6.4).

4.5 Testing

Similarity texter has been tested under the following operating systems, and web browsers:

Ubuntu 14.04 (a GNU/Linux distribution)	**Microsoft Windows 7**
Google Chrome (v48.0)	Google Chrome (v48.0)
Mozilla Firefox (v44.0)	Mozilla Firefox (v44.0)
	Internet Explorer (v11.0)

The web application performs well under the specified versions of the web browsers on both operating systems. However, given the known incompatibility issues as regards the implementation of each web browser, performance rates or accuracy of the output vary among them. The following table lists the major functionalities of the web application, and how they perform under each one of the web browsers specified above.

Google Chrome v48.0	**Mozilla Firefox v44.0**	**Internet Explorer v11.0**
Web storage		
Supported (offline & online)	Supported (offline & online)	Supported only online
Comparison rates		
Good	Best	Good
Animation effects (e.g. auto-scrolling functionality)		
Best	Good	Average
PDF generation rates		
Good	Best	Not tested
PDF layout accuracy		
Missing page break after p. 1	Best	Missing page break after p. 1. Wrapping of line breaks

It should be noted that the proper functionality of the web application cannot be guaranteed on previous versions, than those stipulated above, of the web browsers.

4.5.1 Stress tests

To measure the efficiency, in terms of running time, of the comparison algorithm, as it has been implemented in *similarity texter*, the following stress tests have been carried out. These tests report the time lapse required, in milliseconds, for the completion of the comparison operation, and the repainting of the DOM.

The tables below describe the specifications of the test systems, as well as the text samples used for the testing.

TEST SYSTEMS

Hardware	Windows Duo	Ubuntu Duo	Ubuntu i7
Processor	Intel Core 2 Duo 6300 1,86 GHz (2-core)	Intel Core 2 Duo 6300 1,86 GHz (2-core)	Intel Core i7-2620M 2,7 GHz (4-core)
RAM	3 GiB	3 GiB	8 GiB
Operating system	Windows 7	Ubuntu 14.04	Ubuntu 14.04

TEST INPUT FILES

	Input file 1	Input file 2
Title	Ulysses	The Odyssey of Homer
Author	James Joyce	Homer (Translation: W. Cowper)
Size	1,5 MB	803 kB
Characters	1.573.047	820.979
Pages (ca.)	828	432
Downloaded from	Project Gutenberg	Project Gutenberg
URL	http://www.gutenberg.org/cache/epub/4300/pg4300.txt	http://www.gutenberg.org/files/24269/24269-0.txt

The definition of each stress test (i.e. comparison options), together with the results recorded, follow below.

STRESS TEST 1

Comparison options

Minimum match length	2
Ignore letter case	Enabled
Ignore numbers	Disabled
Ignore punctuation	Enabled
Replace umlaut & ligatures	Enabled

Results (in ms)

System	Google Chrome v48.0	Mozilla Firefox v44.0	Internet Explorer v11.0
Windows Duo	15716,00	15792,99	60609,70 (crashed)
Ubuntu Duo	14839,28	14119,78	-
Ubuntu i7	6269,01	5355,81	-

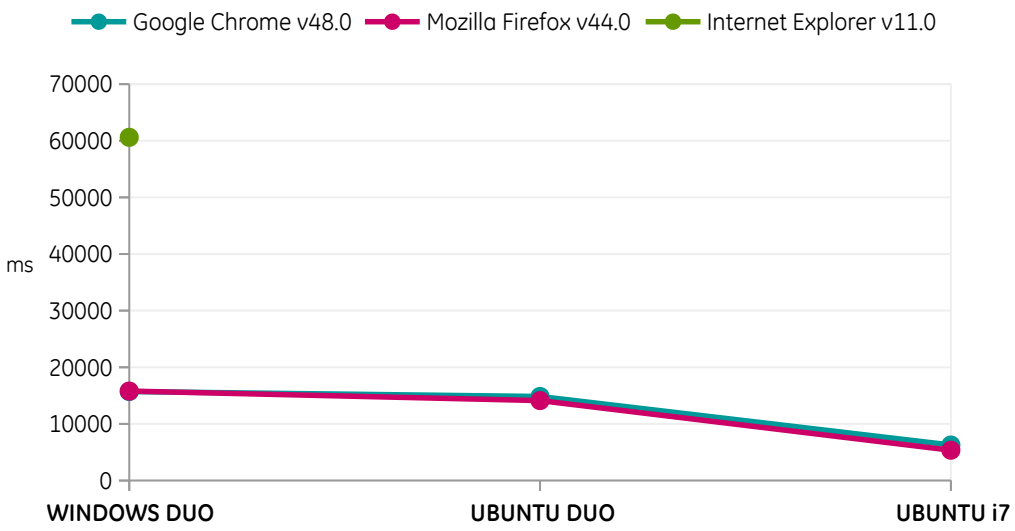

Figure 15. Results' chart of stress test 1

STRESS TEST 2

Comparison options

Minimum match length	4
Ignore letter case	Enabled
Ignore numbers	Disabled
Ignore punctuation	Enabled
Replace umlaut & ligatures	Enabled

Results (in ms)

System	Google Chrome v48.0	Mozilla Firefox v44.0	Internet Explorer v11.0
Windows Duo	13622,00	7110,96	4144,44
Ubuntu Duo	11858,40	8605,44	-
Ubuntu i7	5121,73	3186,63	-

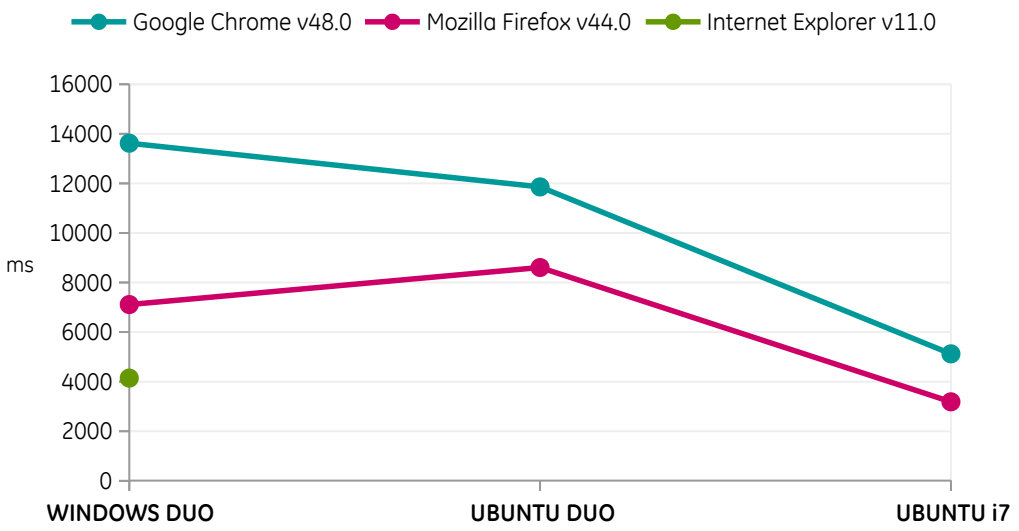

Figure 16. Results' chart of stress test 2

Appendix A: List of Internet home pages

Name	Internet home page
Bootstrap	http://getbootstrap.com/
Bootstrap FileStyle	http://markusslima.github.io/bootstrap-filestyle/
browserify	http://browserify.org/
Copyscape	http://www.copyscape.com/compare.php
flex	http://flex.sourceforge.net/
JPlag	http://jplag.ipd.kit.edu/
jQuery	http://jquery.com/
jsPDF	http://parall.ax/products/jspdf
JSZip	http://stuk.github.io/jszip/
Node.js	http://nodejs.org/en/
PDF24 Creator	http://en.pdf24.org/creator.html
PDFKit	http://pdfkit.org/
pdfmake	http://pdfmake.org/#/
SIM	http://dickgrune.com/Programs/similarity_tester/
Similarity Analyzer	http://tool.motoricerca.info/similarity-analyzer.phtml
String Similarity Test	http://www.tools4noobs.com/online_tools/string_similarity/
Text Comparison	http://people.f4.htw-berlin.de/~weberwu/Tools/Text-Compare.html (English version) http://de.vroniplag.wikia.com/wiki/Quelle:Textvergleich (German version)
WCopyfind	http://plagiarism.bloomfieldmedia.com/z-wordpress/software/wcopyfind/
XRegExp	http://xregexp.com/

Appendix B: Activity diagrams of SIM

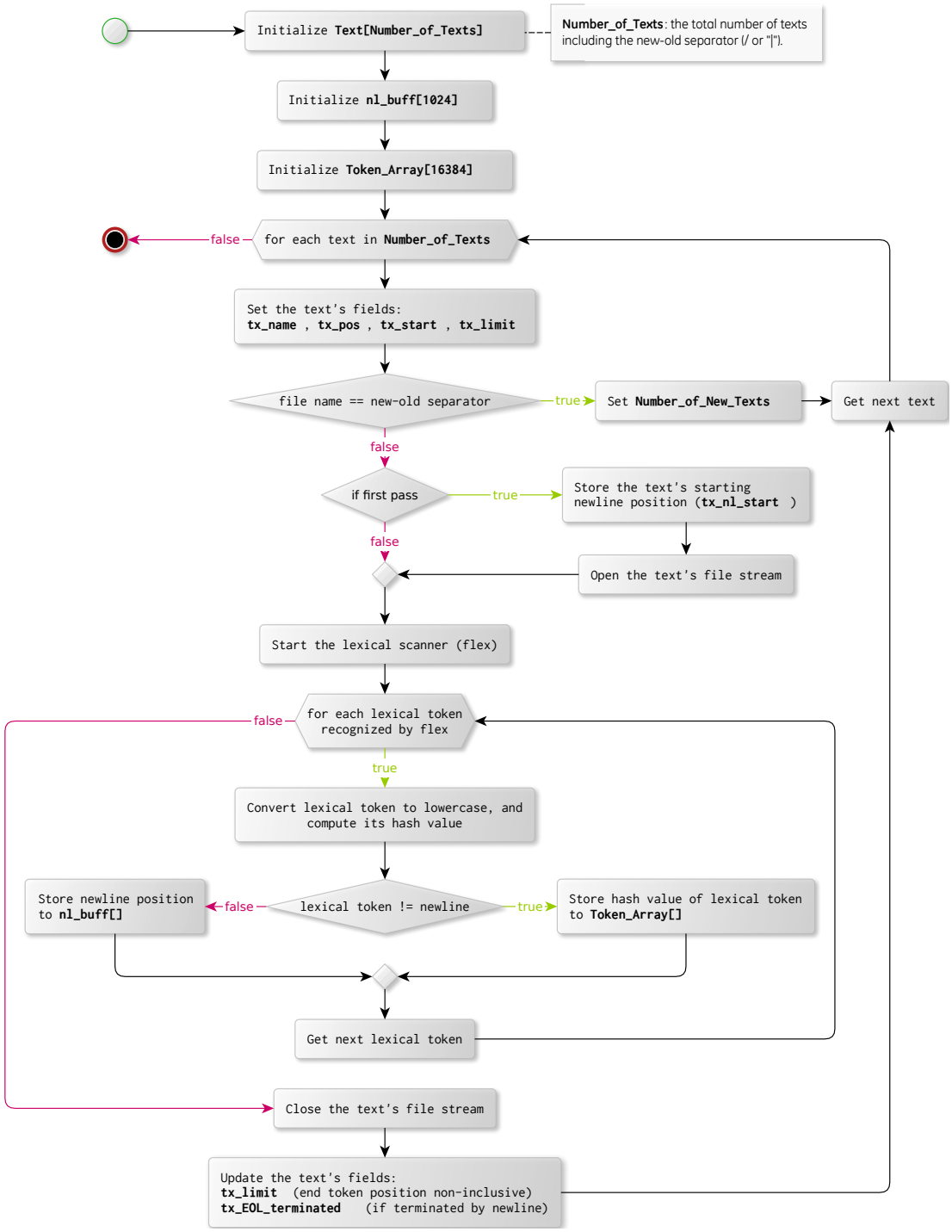

Figure 17. Activity diagram: Read input files

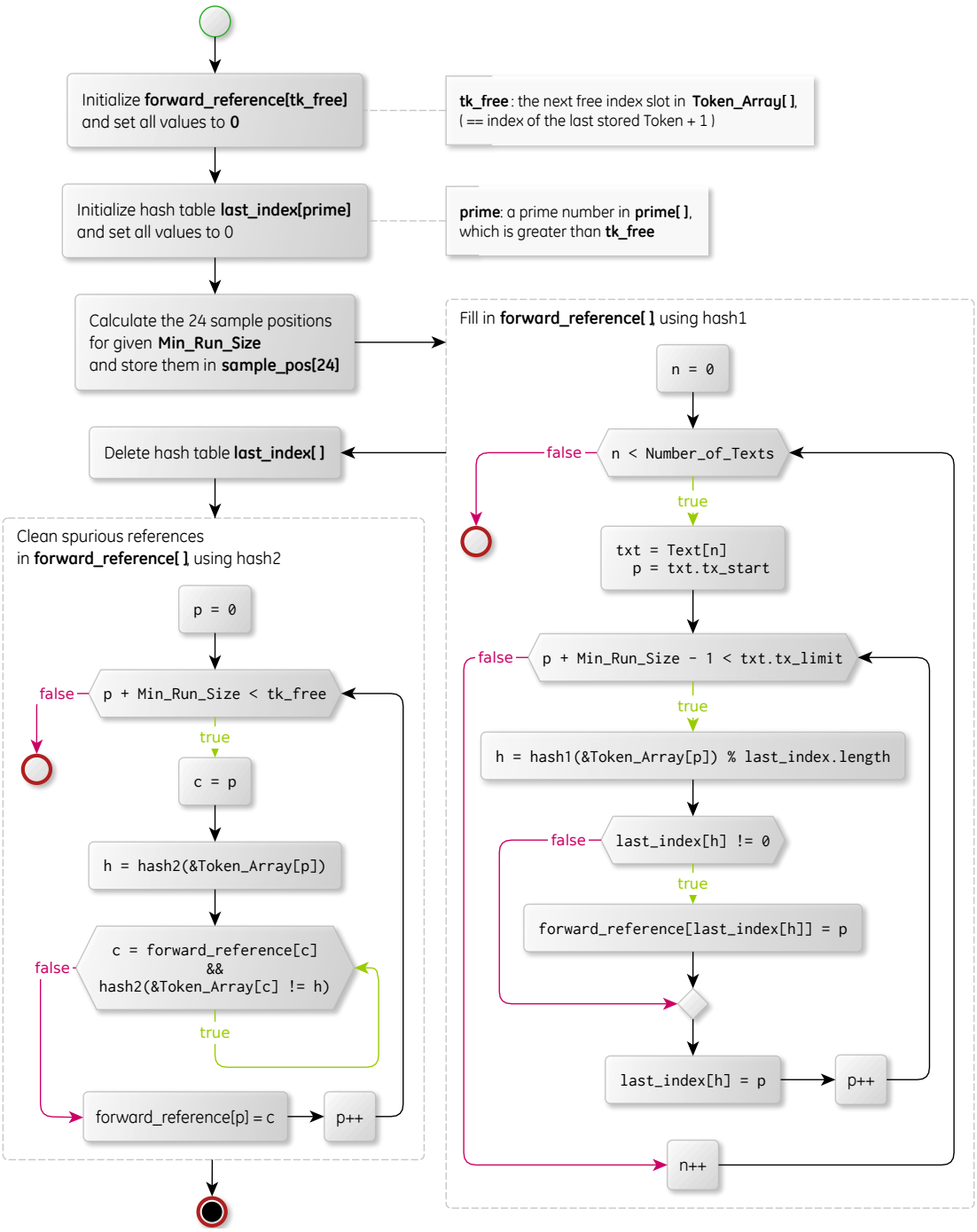

Figure 18. Activity diagram: Build forward reference table

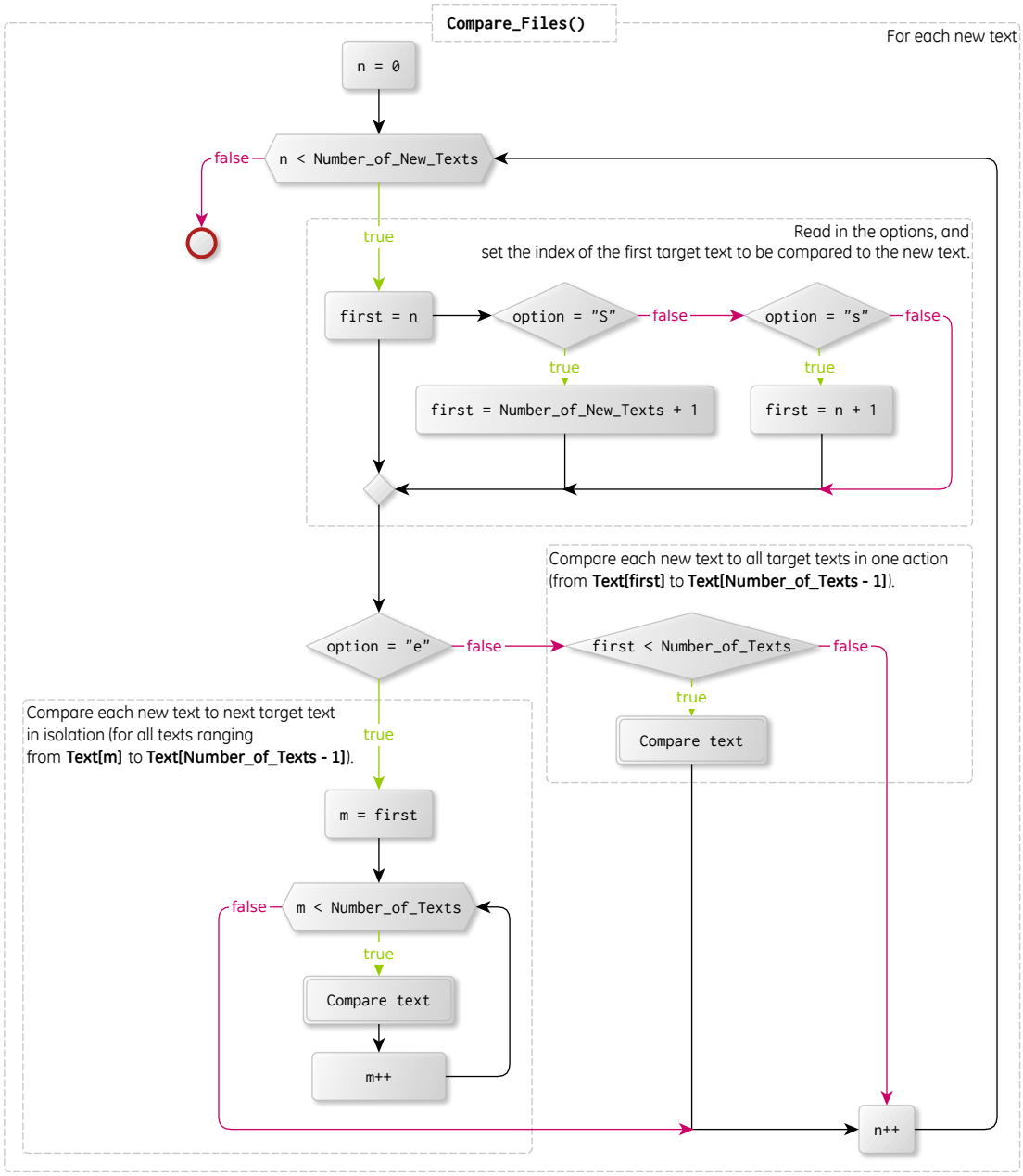

Figure 19. Activity diagram: Compare files (1st part)

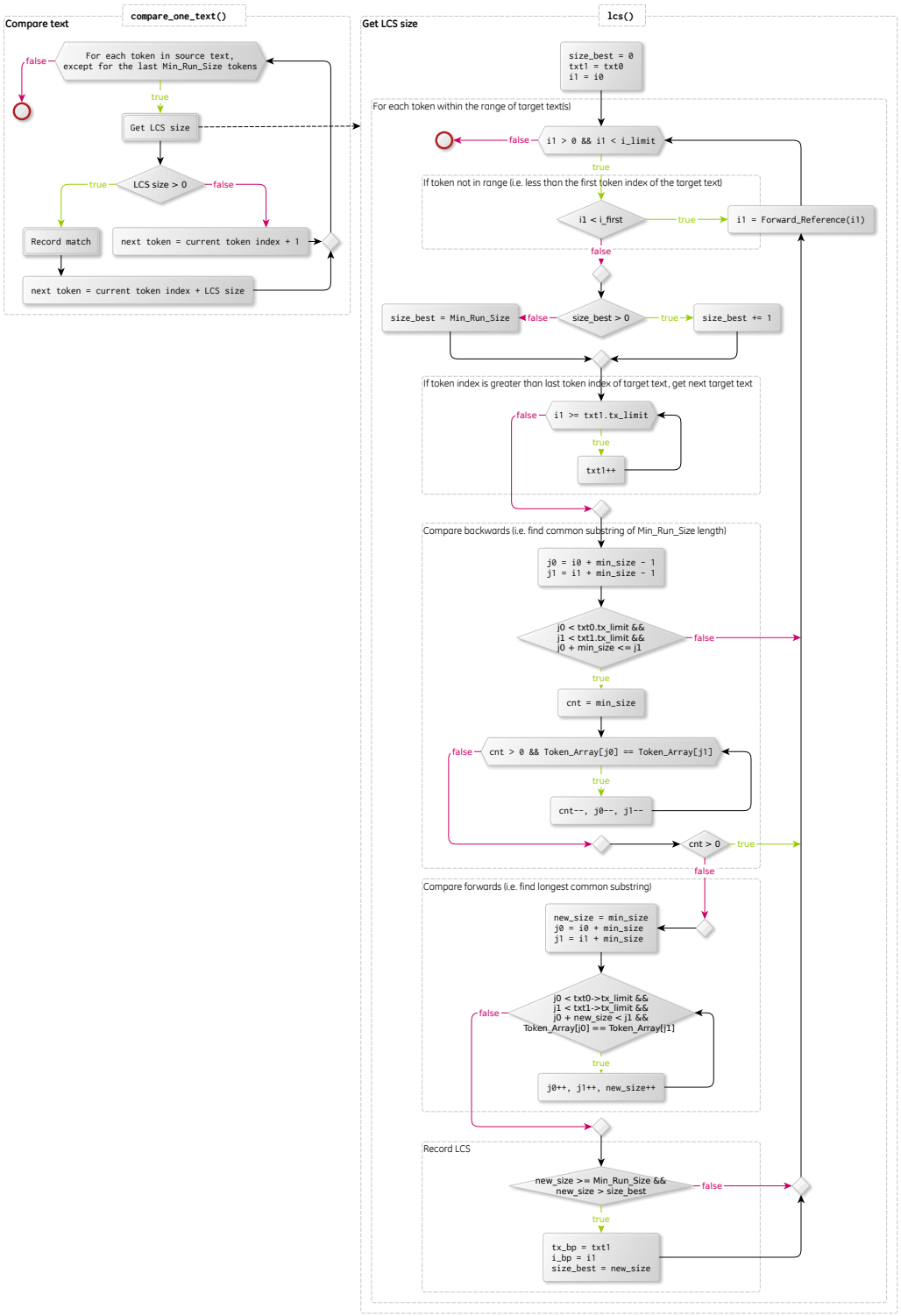

Figure 20. Activity diagram: Compare files (2nd part)

Appendix C: Sequence diagrams of SIM

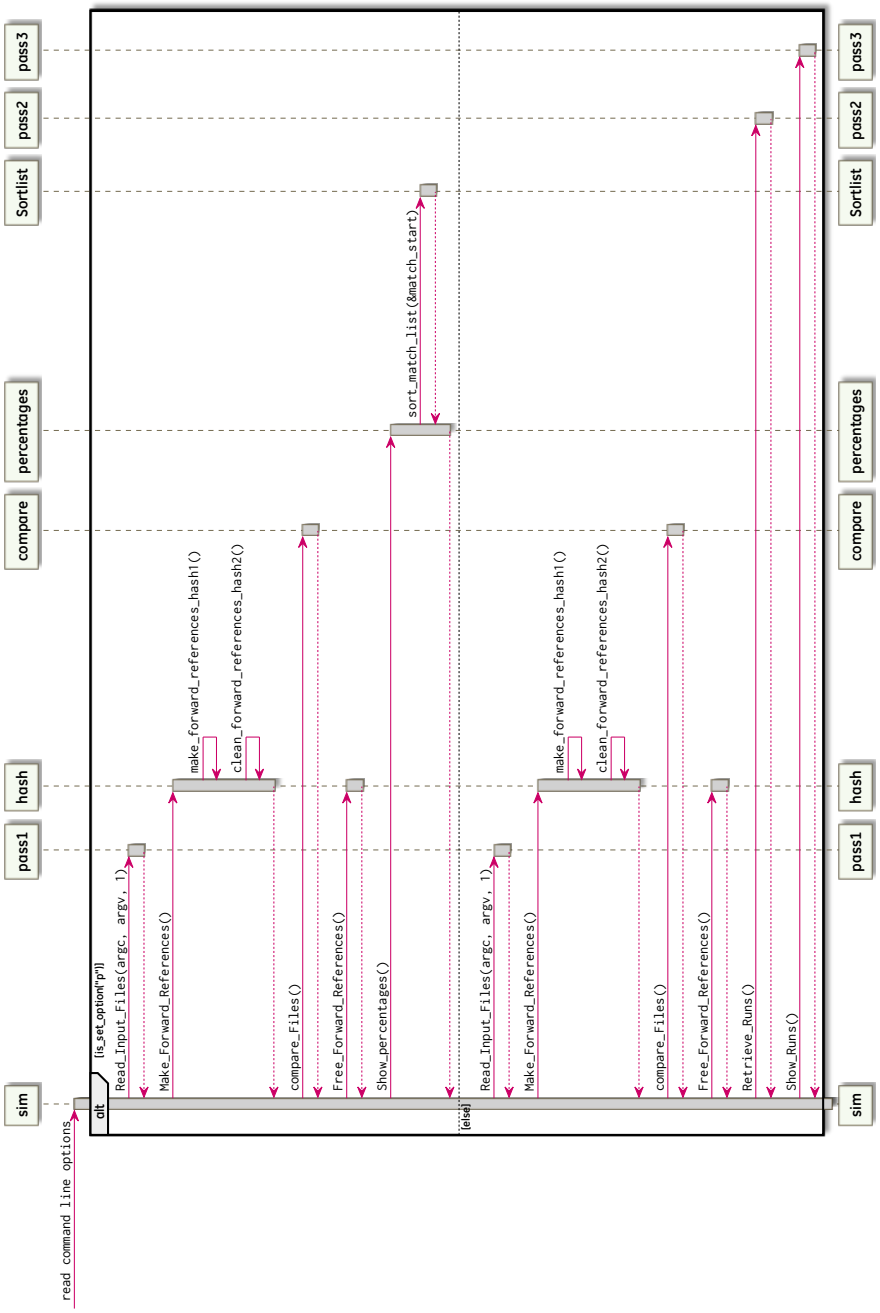

Figure 21. Basic sequence diagram of *sim_text*

Appendix D: File diagrams of SIM

File diagram constitutes an exceptional term that has been intentionally used for the purpose of expressing source code, written in a procedural language such as C, as a UML class diagram. The following diagrams attempt to model each `.c` file, together with its header(s) (`.h`), as a single unit/module.

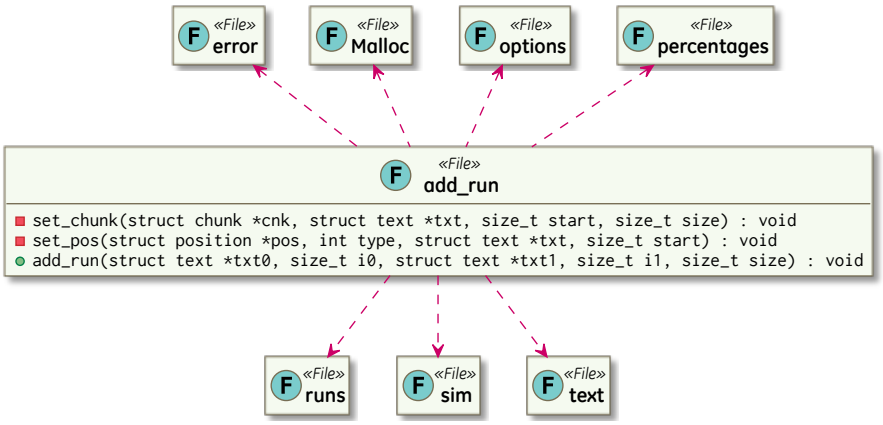

Figure 22. `add_run` file diagram

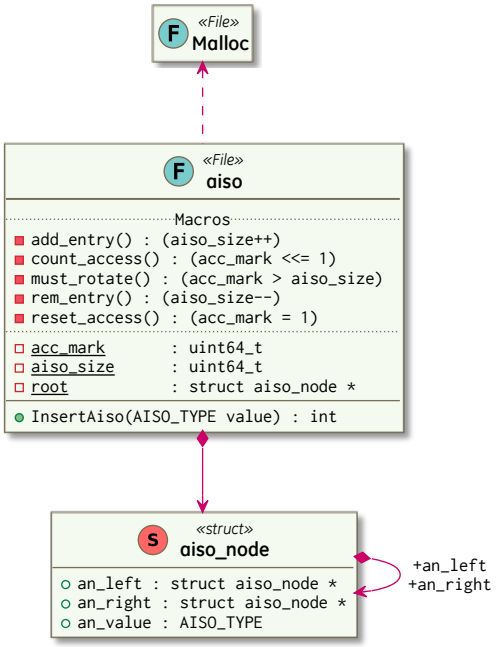

Figure 23. `aiso` file diagram

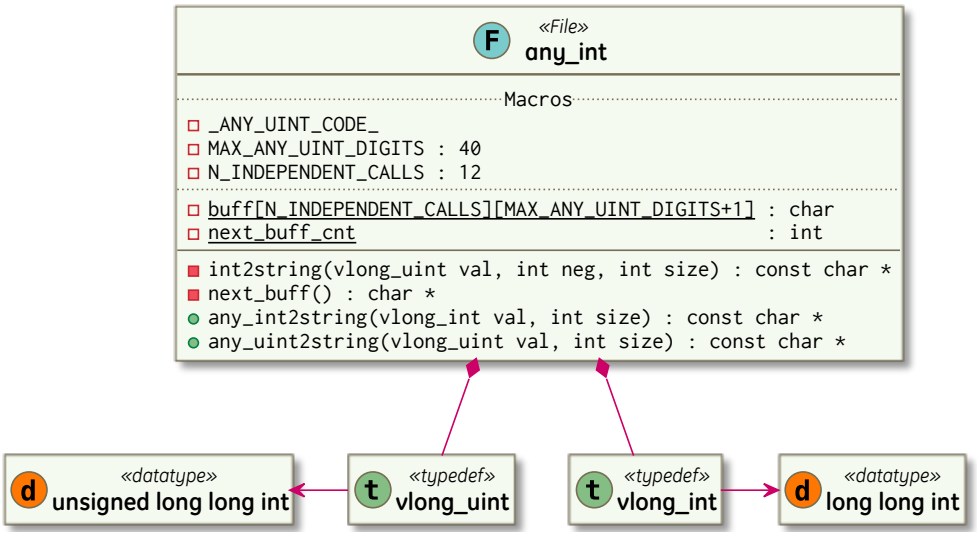

Figure 24. `any_int` file diagram

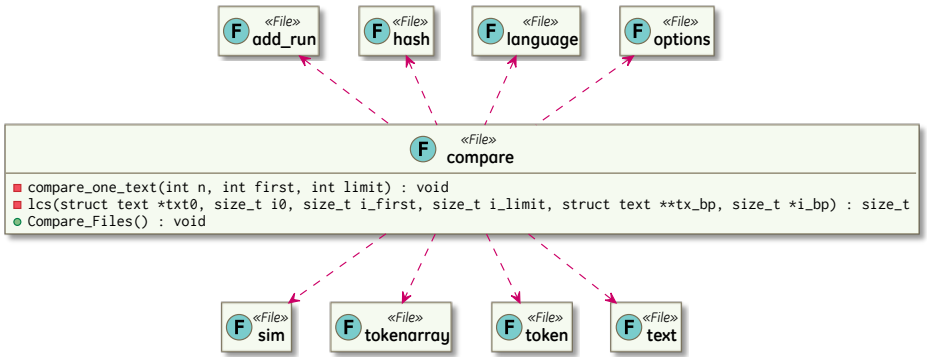

Figure 25. `compare` file diagram

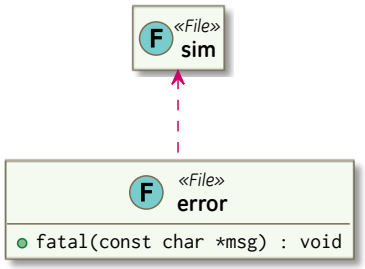

Figure 26. `error` file diagram

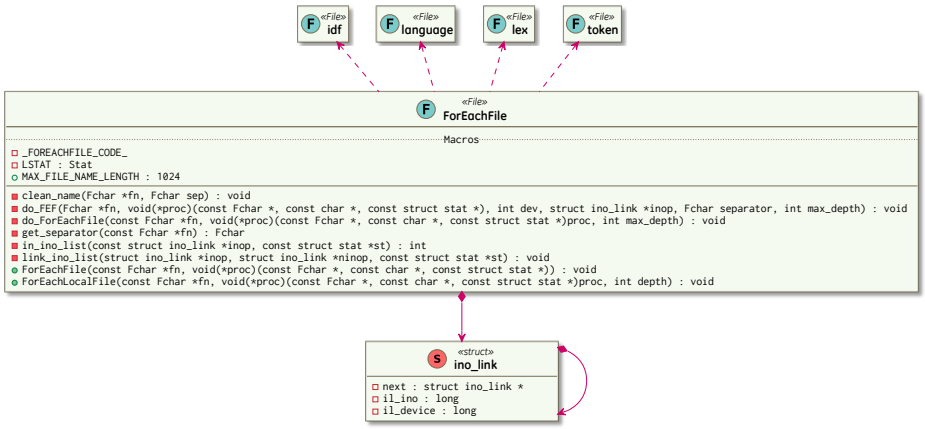

Figure 27. `ForEachFile` file diagram

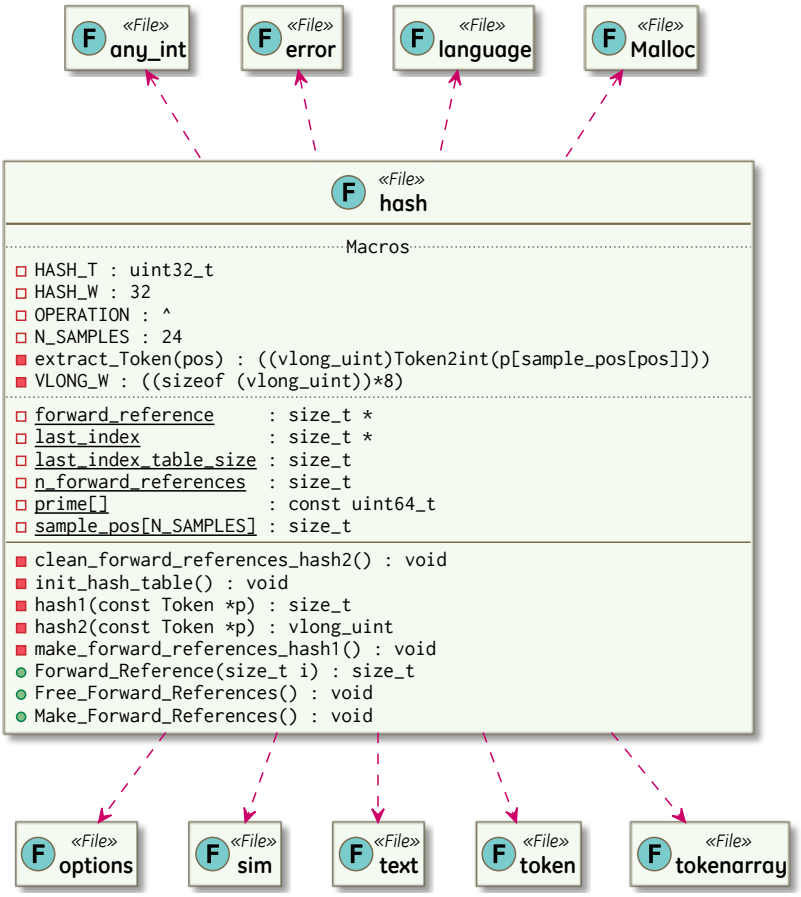

Figure 28. `hash` file diagram

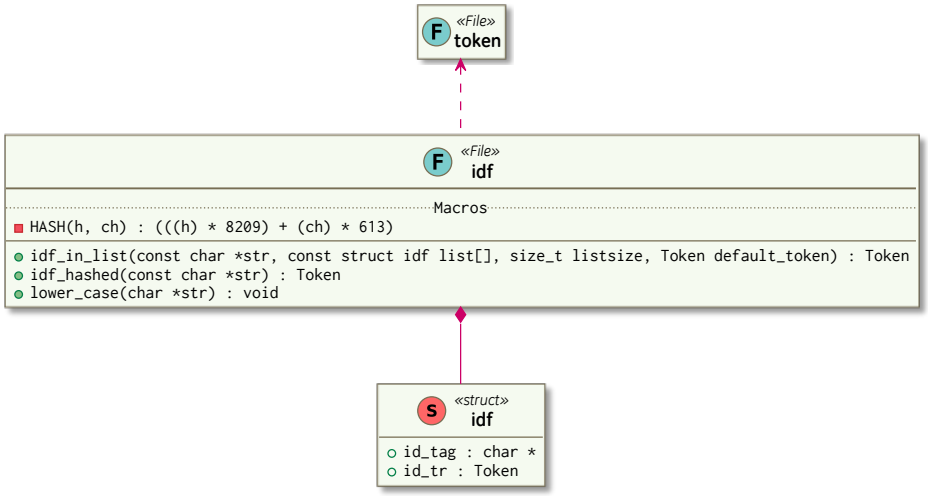

Figure 29. `idf` file diagram

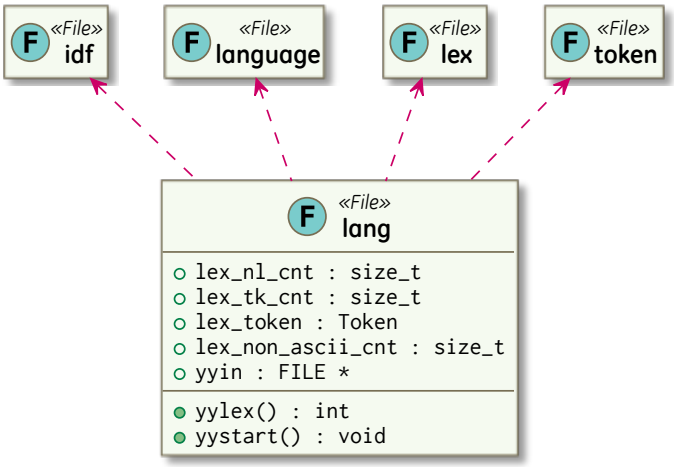

Figure 30. `lang` file diagram

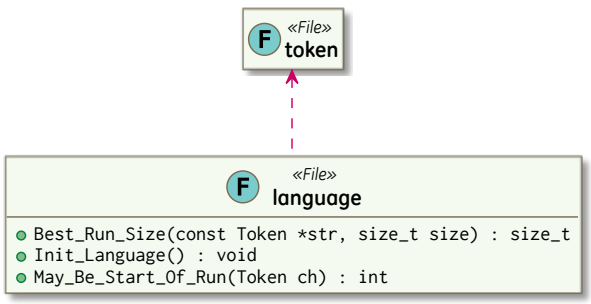

Figure 31. `language` file diagram

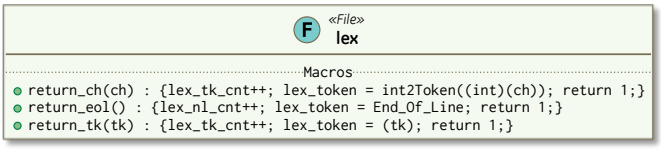

Figure 32. `lex` file diagram

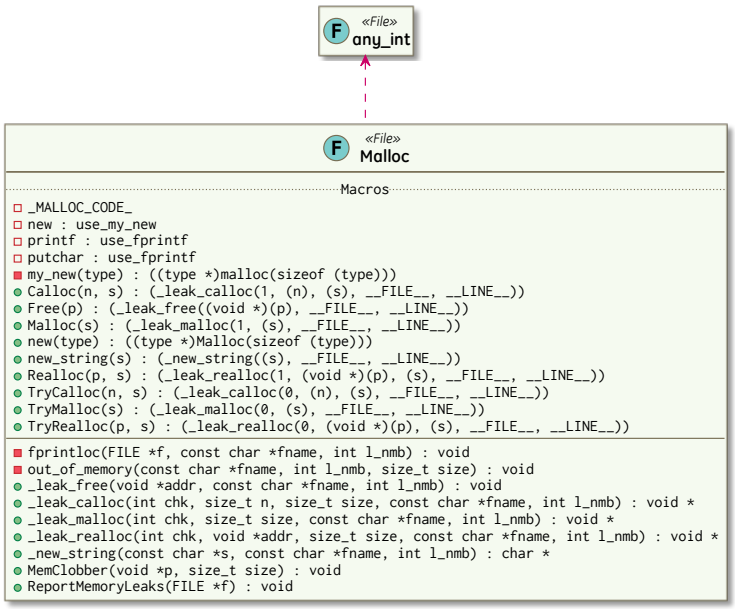

Figure 33. `Malloc` file diagram

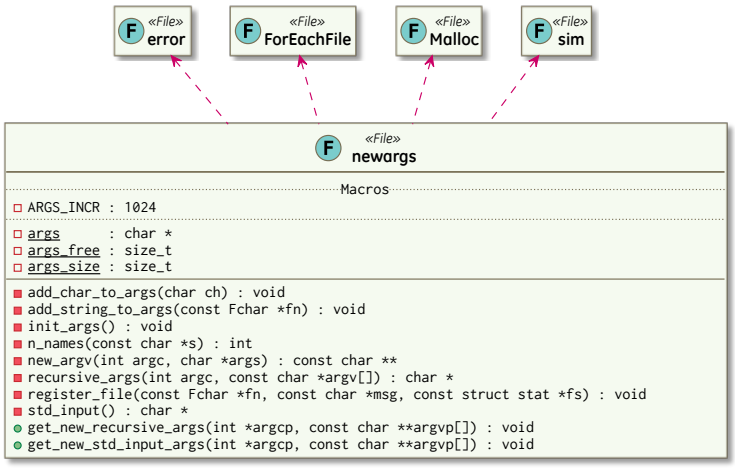

Figure 34. `newargs` file diagram

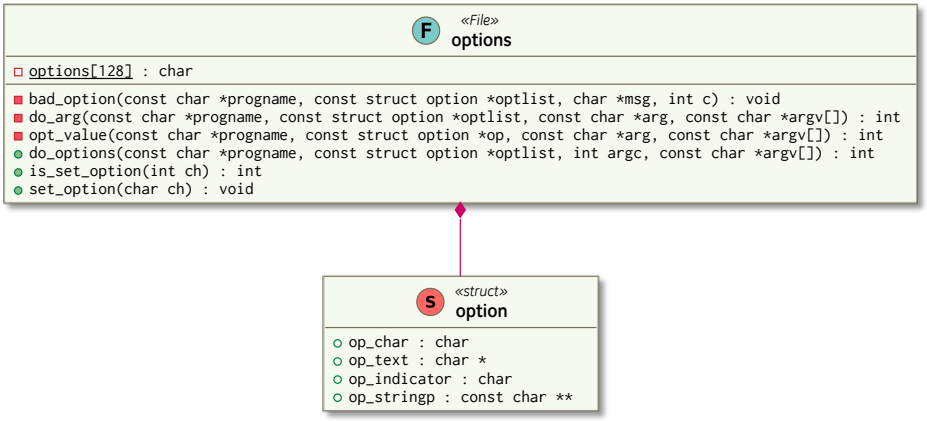

Figure 35. `options` file diagram

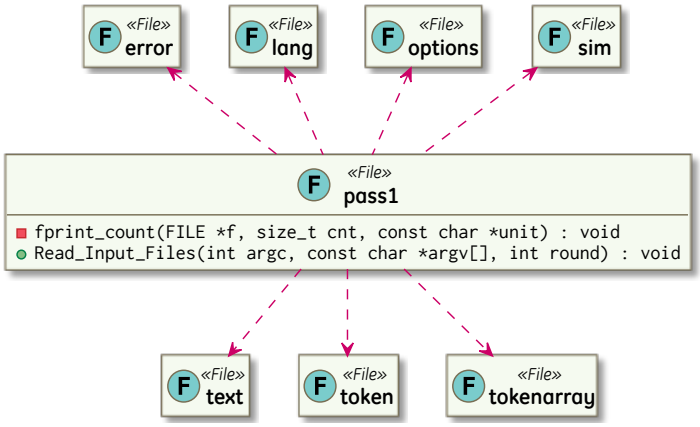

Figure 36. `pass1` file diagram

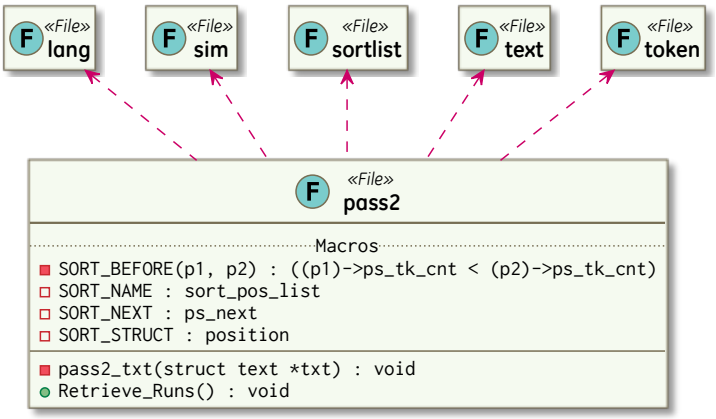

Figure 37. `pass2` file diagram

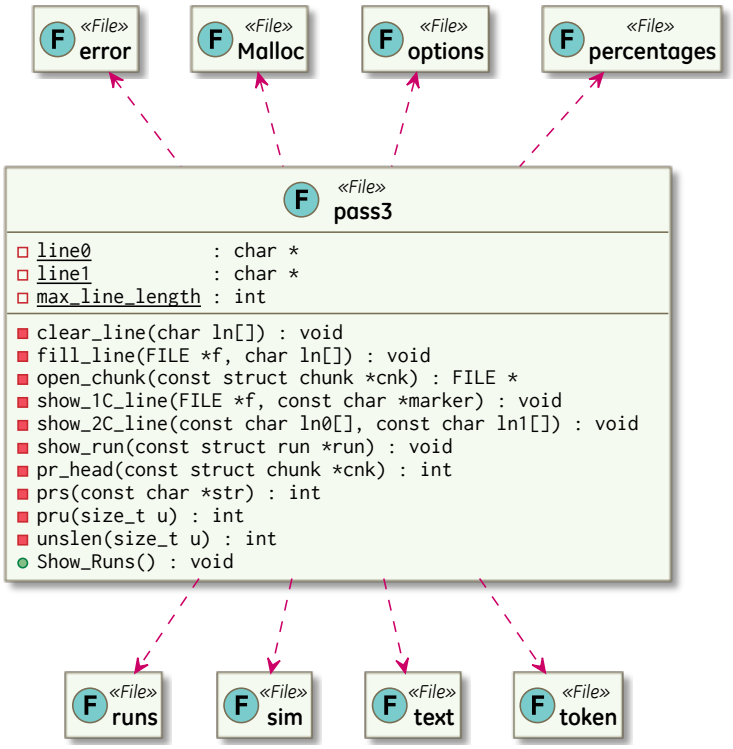

Figure 38. pass3 file diagram

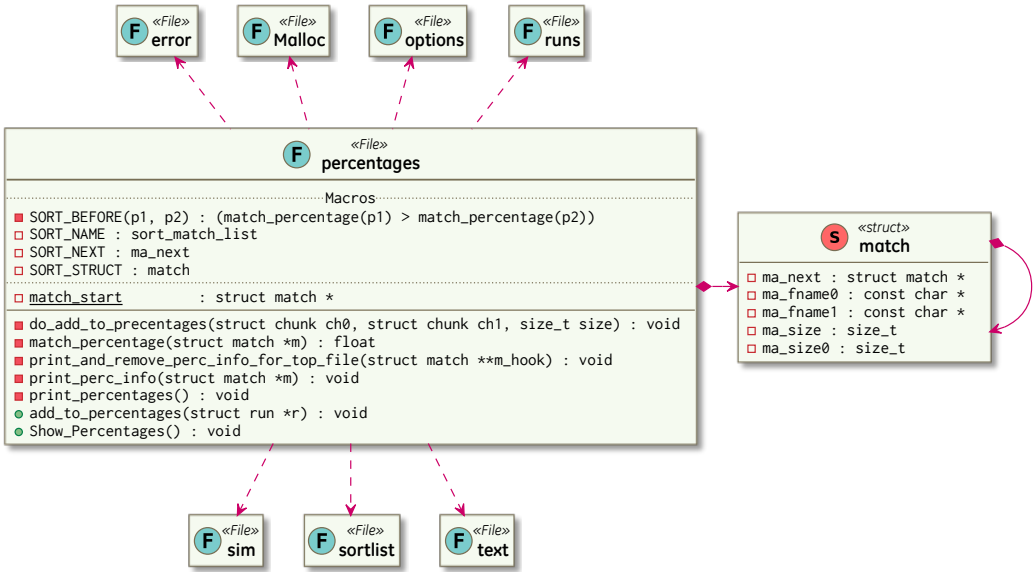

Figure 39. percentages file diagram

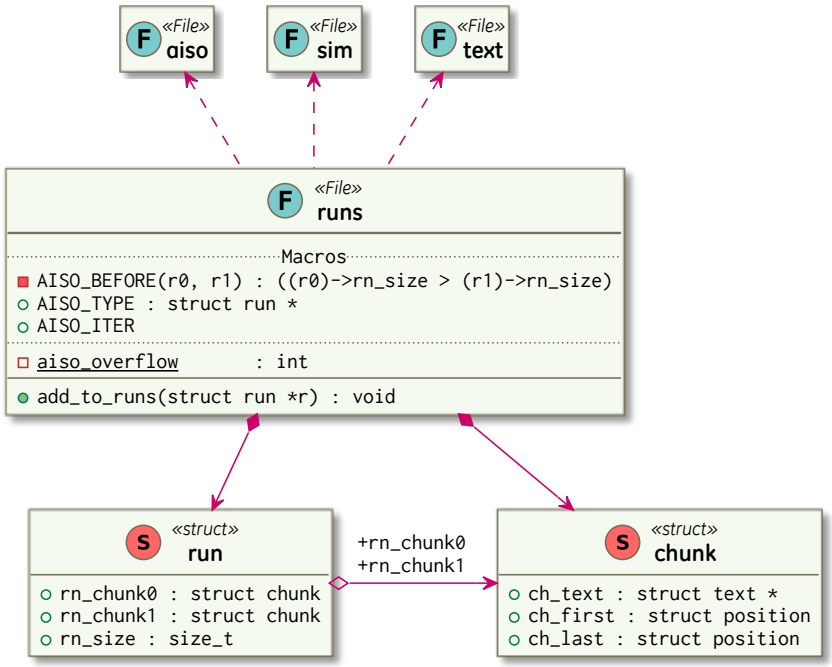

Figure 40. `runs` file diagram

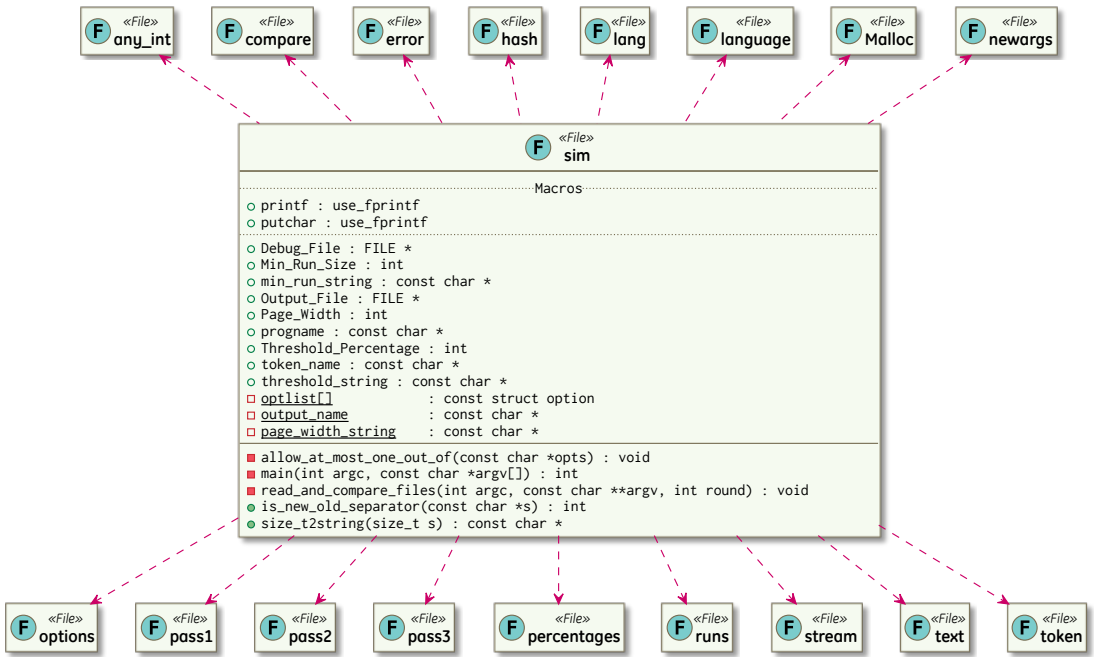

Figure 41. `sim` file diagram

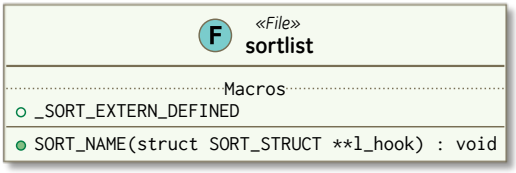

Figure 42. `sortlist` file diagram

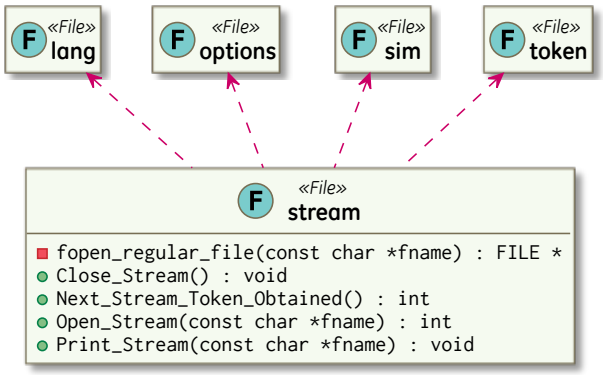

Figure 43. `stream` file diagram

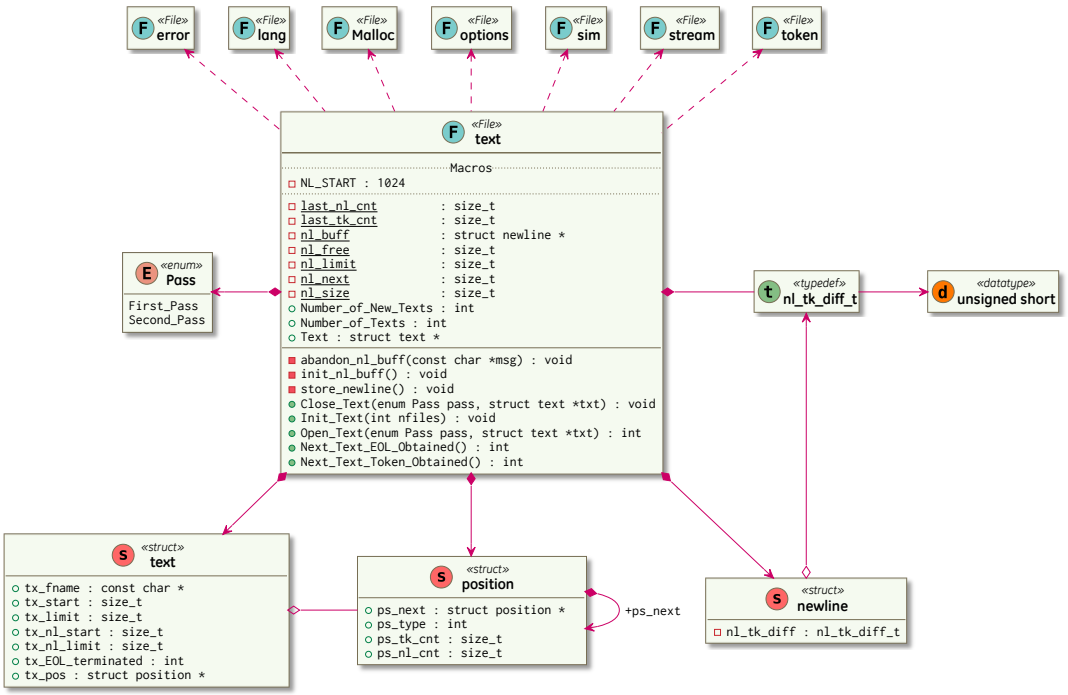

Figure 44. `text` file diagram

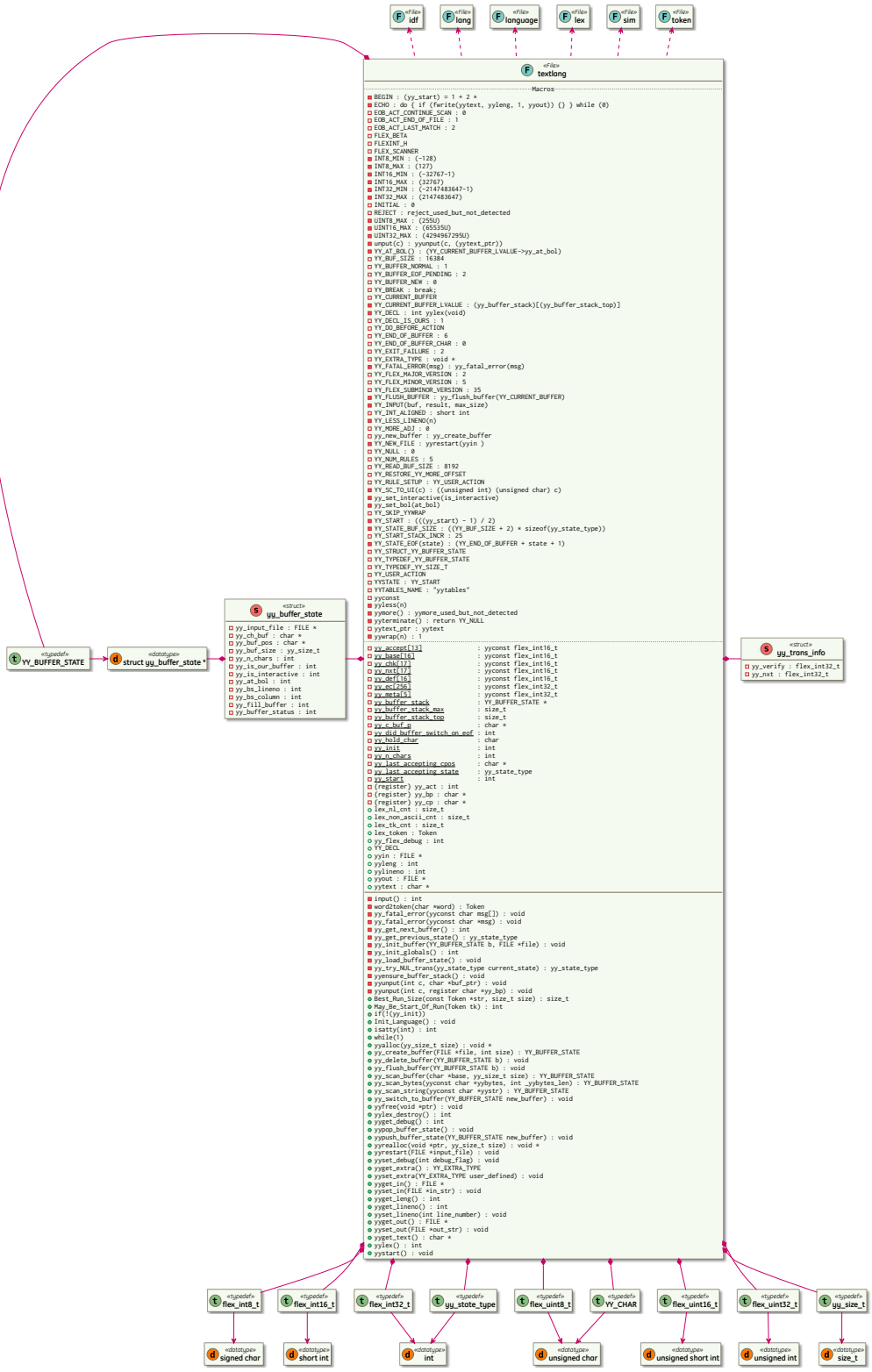

Figure 45. `textlang` file diagram

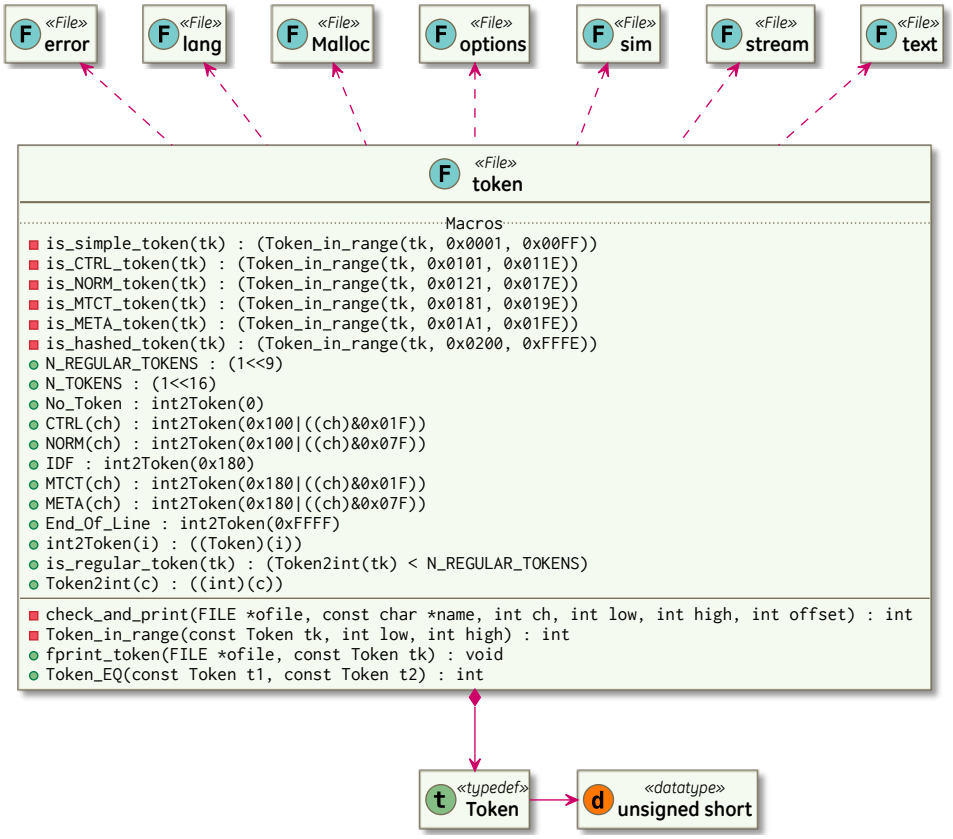

Figure 46. `token` file diagram

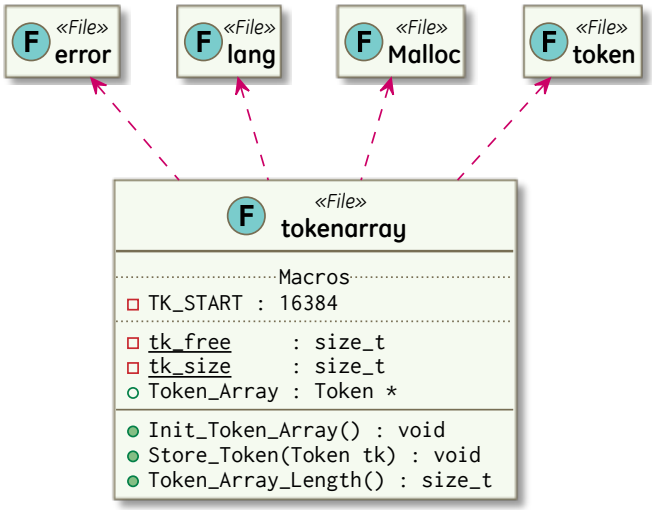

Figure 47. `tokenarray` file diagram

Appendix E: Source code

Source file 5. src/js/main.js

```
/* jshint undef:true, unused:true, node:true, browser:true */
'use strict';

var $   = require('jQuery');
var App = require('./app/app.js');

// Main execution entry point
$(window).load(function() {
  setTimeout(function() {
    $(".loader").addClass('shrinked');
    var app = new App('simtexter');
  }, 700);
});
```

Source file 6. src/js/app/app.js

```
/* jshint undef:true, unused:true, node:true, browser:true */
'use strict';

var Controller = require('./controller.js');
var Storage    = require('./storage.js');
var Template   = require('./template.js');
var View       = require('./view.js');

/**
 * Creates an instance of the application.
 * @constructor
 * @this  {App}
 * @param {String} namespace - the namespace of the app (i.e. "simtexter")
 */
function App(namespace) {
  // App's default settings (comparison & input reading options)
  var defaults = {
      'minMatchLength'    : { id: '#min-match-length',   type: 'inputText',
          value: 4     },
      'ignoreFootnotes'   : { id: '#ignore-footnotes',   type: 'checkbox',
          value: false },
      'ignoreLetterCase'  : { id: '#ignore-letter-case', type: 'checkbox',
          value: true  },
      'ignoreNumbers'     : { id: '#ignore-numbers',     type: 'checkbox',
          value: false },
      'ignorePunctuation' : { id: '#ignore-punctuation', type: 'checkbox',
          value: true  },
      'replaceUmlaut'     : { id: '#replace-umlaut',     type: 'checkbox',
          value: true  }
    };

  this.storage    = new Storage(namespace, defaults);
  this.template   = new Template();
  this.view       = new View(this.template);
  this.controller = new Controller(this.storage, this.view);
}

module.exports = App;
```

Source file 7. src/js/app/controller.js

```
/* jshint undef:true, unused:true, node:true, browser:true */
'use strict';

var $               = require('jQuery');
var FileInputReader = require('../inputReader/fileInputReader.js');
var InputText       = require('./inputText.js');
var SimTexter       = require('../simtexter/simtexter.js');
var TextInputReader = require('../inputReader/textInputReader.js');

/**
 * Creates an instance of a {Controller},
 * which handles user interaction (data reading, input control, comparison).
 * Interacts with the {View} object to render the final output.
 * @constructor
 * @this  {Controller}
 * @param {Storage} storage - the object that holds the app's settings
 * @param {View}    view    - the app's view
 */
function Controller(storage, view) {
  this.storage              = storage;
  this.view                 = view;
  this.maxCharactersPerPage = 1900;
  this.maxNumberOfPages     = 500;
  this.inputTexts           = [ new InputText(), new InputText() ];

  this._bindEvents();
  this._updateUI(this.storage.data);
}

/**
 * Displays a warning message if input is too long (> maxNumberOfPages).
 * @function
 * @private
 * @param {Number} idx - the index of the {InputText} object in inputTexts[]
 */
Controller.prototype._alertLongInput = function(idx) {
  var self = this;

  // Compute approximate number of pages for inputText
  var nrOfPages = self.inputTexts[idx].getNumberOfPages(self.
      maxCharactersPerPage);
  // If greater than maximum number of pages, display warning message
  if (nrOfPages > self.maxNumberOfPages) {
    var inputMode = self.inputTexts[idx].mode;
    var message = [
        inputMode, ' ', (idx + 1), ' is too long. To prevent visualization
            issues, please consider truncating this ', inputMode.toLowerCase(),
            '.'
      ].join('');
    var delay = self._computeReadingSpeed(message);
    self.view.showAlertMessage('warning', message, delay);
  }
};

/**
 * Binds events.
 * @function
 * @private
 */
Controller.prototype._bindEvents = function() {
  var self = this;

```

```
60    self.view.bind('changeSpinnerInput', function(id, newValue) {
61      self._updateStorage(id, newValue);
62    });
63
64    self.view.bind('compare', function() {
65      self._compare();
66    });
67
68    self.view.bind('dismissAlert');
69    self.view.bind('hidePrintDialog');
70    self.view.bind('initBootstrap');
71
72    self.view.bind('inputFile', function(file, idx, loadingElem, tabPaneId) {
73      self._readFile(file, idx, loadingElem, tabPaneId);
74    });
75
76    self.view.bind('inputText', function(text, idx, tabPaneId) {
77      self._readText(text, idx, tabPaneId);
78    });
79
80    self.view.bind('print', function(hideModalPromise) {
81      self._print(hideModalPromise);
82    });
83
84    self.view.bind('resize');
85    self.view.bind('scrollToMatch');
86    self.view.bind('selectTab');
87
88    self.view.bind('selectHTMLOption', function(idx, newValue, text) {
89      self.inputTexts[idx].setHTMLOption(newValue);
90      if (text) {
91        self._readText(text, idx, self.inputTexts[idx].tabPaneId);
92      }
93    });
94
95    self.view.bind('selectSettingsOption', function(id, newValue) {
96      self._updateStorage(id, newValue);
97    });
98
99    self.view.bind('showPrintDialog');
100   self.view.bind('toggleInputPanel');
101   self.view.bind('toggleSettingsSidebar');
102   self.view.bind('toggleSettingsSidebarPanes');
103 };
104
105 /**
106  * Initiates the comparison process.
107  * @function
108  * @private
109  */
110 Controller.prototype._compare = function() {
111   var self = this;
112
113   if (self._isInputValid()) {
114     self.view.toggleWaitingCursor('show');
115     var simtexter = new SimTexter(self.storage);
116
117     setTimeout(function() {
118       simtexter.compare(self.inputTexts).then(
119         // On success, update information nodes and display similarities
120         function(nodes) {
121           self.view.results = {
122             texts         : simtexter.texts,
123             uniqueMatches : simtexter.uniqueMatches
```

```
124           };
125
126           self.view.createTemplates();
127           self.view.showSimilarities(nodes);
128           self.view.resetScrollbars();
129         },
130         // On error, clear output panel and display warning message
131         function(message) {
132           self.view.clearOutputPanel();
133           var delay = self._computeReadingSpeed(message);
134           self.view.showAlertMessage('info', message, delay);
135         }
136       );
137     }, 200);
138   }
139 };
140
141 /**
142  * Returns the amount of time in milliseconds
143  * that a user needs in order to read a message.
144  * @function
145  * @private
146  * @param {String} message - the message to be read
147  */
148 Controller.prototype._computeReadingSpeed = function(message) {
149   var minMS = 6000;
150   var speed = Math.round(message.length / 40) * 4000;
151   return (speed > minMS) ? speed : minMS;
152 };
153
154 /**
155  * Checks if the user has provided a valid input
156  * in both source and target input panes.
157  * If not, the user is prompted.
158  * @function
159  * @private
160  * @returns {Boolean} - true if input is valid, else false.
161  */
162 Controller.prototype._isInputValid = function() {
163   var self = this,
164       isValid = true,
165       activeTabPaneIds = self.view.getActiveTabPaneIds(),
166       iTextsLength = self.inputTexts.length;
167
168   for (var i = 0; i < iTextsLength; i++) {
169     var inputText = self.inputTexts[i];
170     var activeTabPaneId = activeTabPaneIds[i];
171
172     var isInputTextValid = (inputText.text !== undefined && inputText.tabPaneId
          === activeTabPaneId);
173
174     if (!isInputTextValid) {
175       self.view.toggleErrorStatus('show', activeTabPaneId);
176     } else {
177       self.view.toggleErrorStatus('hide', activeTabPaneId);
178     }
179
180     isValid = isValid && isInputTextValid;
181   }
182
183   return isValid;
184 };
185
186 /**
```

```
187  * Sends the contents of the current window
188  * to the system's printer for printing.
189  * @function
190  * @private
191  * @param {Promise} hideModalPromise - a promise that handles the hiding
192  *                                     of the 'PRINT OUTPUT' dialog.
193  *                                     When resolved, the current window
194  *                                     is sent to printing.
195  */
196 Controller.prototype._print = function(hideModalPromise) {
197   var success = function() {
198     setTimeout(function() {
199       window.print();
200     }, 700);
201   };
202
203   $.when(hideModalPromise).then(success);
204 };
205
206 /**
207  * Extracts the contents of the selected file
208  * and updates the relevant fields of the {InputText} object.
209  * @function
210  * @private
211  * @param {FileList} file        - the file selected by the user
212  * @param {Number}   idx         - the index of the {InputText} object
213  *                                 in inputTexts[] to be updated.
214  *                                 0: input in left-side pane
215  *                                 1: input in right-side pane
216  * @param {Object}   loadingElem - the node element that shows
217  *                                 the progress of reading
218  * @param {String}   tabPaneId   - the id of the active tab pane
219  */
220 Controller.prototype._readFile = function(file, idx, loadingElem, tabPaneId) {
221   var self = this,
222       ignoreFootnotes = self.storage.getItemValueByKey('ignoreFootnotes');
223
224   var success = function(text) {
225       // Update {InputText} object
226       self.inputTexts[idx].setFileInput(file, text, tabPaneId);
227       self.view.loading('done', loadingElem);
228       self.view.clearTabPaneTextInput(idx);
229       self._alertLongInput(idx);
230     };
231
232     var error = function(message) {
233       self.inputTexts[idx].reset();
234       self.view.loading('error', loadingElem);
235       self.view.clearTabPaneTextInput(idx);
236
237       var delay = self._computeReadingSpeed(message);
238       self.view.showAlertMessage('error', message, delay);
239     };
240
241   if (file) {
242     var loadingStarted = self.view.loading('start', loadingElem);
243     var fileInputReader = new FileInputReader(file, ignoreFootnotes);
244     fileInputReader.readFileInput(loadingStarted).then(success, error);
245   } else {
246     self.view.loading('cancel', loadingElem);
247     self.inputTexts[idx].reset();
248   }
249 };
250
```

```
251 /**
252  * Extracts the contents of the typed/pasted HTML/plain text
253  * and updates the relevant fields of the {InputText} object.
254  * @function
255  * @private
256  * @param {String} text      - the HTML/plain text provided by the user
257  * @param {Number} idx       - the index of the {InputText} object
258  *                             in inputTexts[] to be updated.
259  *                             0: input in left-side pane,
260  *                             1: input in right-side pane
261  * @param {String} tabPaneId - the id of the active tab pane
262  */
263 Controller.prototype._readText = function(text, idx, tabPaneId) {
264   var self = this;
265
266   var success = function(cleanedText) {
267     // Update {InputText} object
268     self.inputTexts[idx].setTextInput(cleanedText, tabPaneId);
269     self.view.toggleCompareBtn('enable');
270     self.view.clearTabPaneFileInput(idx);
271     self._alertLongInput(idx);
272   };
273
274   var error = function(message) {
275     self.inputTexts[idx].reset();
276     self.view.toggleCompareBtn('enable');
277     var delay = self._computeReadingSpeed(message);
278     self.view.showAlertMessage('error', message, delay);
279   };
280
281   if (text.length > 0 && /\S/.test(text)) {
282     if (self.inputTexts[idx].isHTML) {
283       self.view.toggleCompareBtn('disable');
284       var textInputReader = new TextInputReader();
285       textInputReader.readTextInput(text).then(success, error);
286     } else {
287       success(text);
288     }
289   } else {
290     self.inputTexts[idx].reset();
291   }
292 };
293
294 /**
295  * Updates the value of a setting, stored in the {Storage} object.
296  * @function
297  * @private
298  * @param {String}           id       - the id of the setting
299  * @param {(Boolean|Number)} newValue - the new value of the setting
300  */
301 Controller.prototype._updateStorage = function(id, newValue) {
302   var self = this;
303   self.storage.setItemValueById(id, newValue);
304 };
305
306 /**
307  * Updates the {View} object with the values of the settings,
308  * stored in the {Storage} object.
309  * @function
310  * @private
311  * @param {Object} data - the object that holds the storage's settings
312  */
313 Controller.prototype._updateUI = function(data) {
314   var self = this;
```

```
315
316   for (var key in data) {
317     var obj = data[key];
318     self.view.updateUIOption(obj.id, obj.type, obj.value);
319   }
320 };
321
322 module.exports = Controller;
```

Source file 8. src/js/app/inputText.js

```
1  /* jshint undef:true, unused:true, node:true, browser:true */
2  'use strict';
3
4  /**
5   * Creates an instance of a {InputText},
6   * which holds information on the user input.
7   * @constructor
8   * @this  {InputText}
9   * @param {String} mode      - the mode of input (i.e. "file" or "text")
10  * @param {File}   file      - the file selected by the user
11  * @param {String} text      - the input string
12  * @param {String} tabPaneId - the id of the tab pane
13  */
14 function InputText(mode, file, text, tabPaneId) {
15   this.tabPaneId  = tabPaneId;
16   this.mode       = mode;
17   this.isHTML     = false;
18   this.fileName   = (file && file.name);
19   this.text       = text;
20 }
21
22 /**
23  * Returns the approximate number of pages of the input string.
24  * @function
25  * @param   {Number} maxCharactersPerPage - the maximum number of characters
26  *                                          per page
27  * @returns {Number}                     - the ca. number of pages
28  */
29 InputText.prototype.getNumberOfPages = function(maxCharactersPerPage) {
30   return (this.text.length / maxCharactersPerPage);
31 };
32
33 /**
34  * Resets some fields of the {InputText}.
35  * @function
36  */
37 InputText.prototype.reset = function() {
38   this.tabPaneId  = undefined;
39   this.mode       = undefined;
40   this.fileName   = undefined;
41   this.text       = undefined;
42 };
43
44 /**
45  * Sets the fields for the file input.
46  * @function
47  * @param {File}   file      - the file selected by the user
48  * @param {String} text      - the file input string
49  * @param {String} tabPaneId - the id of the tab pane
50  */
51 InputText.prototype.setFileInput = function(file, text, tabPaneId) {
52   this.tabPaneId  = tabPaneId;
```

```
53    this.mode       = 'File';
54    this.fileName   = file.name;
55    this.text       = text;
56  };
57
58  /**
59   * Sets the fields for the text input.
60   * @function
61   * @param {String} text      - the text input string
62   * @param {String} tabPaneId - the id of the tab pane
63   */
64  InputText.prototype.setTextInput = function(text, tabPaneId) {
65    this.tabPaneId  = tabPaneId;
66    this.mode       = 'Text';
67    this.fileName   = (this.isHTML) ? 'HTML text input' : 'Plain text input';
68    this.text       = text;
69  };
70
71  InputText.prototype.setHTMLOption = function(newValue) {
72    this.isHTML = newValue;
73  };
74
75  module.exports = InputText;
```

Source file 9. src/js/app/storage.js

```
1   /* jshint undef:true, unused:true, node:true, browser:true */
2   'use strict';
3
4   /**
5    * Creates an instance of a {Storage},
6    * which stores the values of the app's settings.
7    * If local storage is supported by the browser,
8    * these settings are also stored under the specified namespace,
9    * thus providing the app with a state.
10   * The last stored settings will be restored
11   * when refreshing the page or restarting the browser.
12   * @constructor
13   * @this  {Storage}
14   * @param {String} namespace - the namespace of the app (i.e. "simtexter")
15   * @param {Object} data      - the object that holds the app's settings
16   */
17  function Storage(namespace, data) {
18     this._db  = namespace;
19     this.data = this._initialize(namespace, data);
20  }
21
22  /**
23   * Returns the value of a setting, retrieved by its key value.
24   * @function
25   * @param {String} key - the key value of the setting
26   */
27  Storage.prototype.getItemValueByKey = function(key) {
28    var self = this;
29    return self._getItemByKey(key).value;
30  };
31
32  /**
33   * Sets the new value of a setting, retrieved by its id value.
34   * @function
35   * @param {String}           id       - the id of the setting
36   * @param {(Boolean|Number)} newValue - the new value of the setting
37   */
```

```
38 Storage.prototype.setItemValueById = function(id, newValue) {
39   var self = this,
40       item = self._getItemById(id);
41
42   item.value = newValue;
43   self._save(self.data);
44 };
45
46 /**
47  * Retrieves a setting by its id value.
48  * @function
49  * @private
50  * @param {String} id - the id of the setting
51  */
52 Storage.prototype._getItemById = function(id) {
53   var self = this,
54       data = self.data;
55
56   for (var key in data) {
57     var obj = data[key];
58     if (obj.id === id) {
59       return obj;
60     }
61   }
62
63   return undefined;
64 };
65
66 /**
67  * Retrieves a setting by its key value.
68  * @function
69  * @private
70  * @param {String} key - the key value of the setting
71  */
72 Storage.prototype._getItemByKey = function(key) {
73   var self = this;
74   return self.data[key];
75 };
76
77 /**
78  * Stores the app's settings in the web browser's local storage
79  * under the specified namespace.
80  * If local storage is not supported, stores the settings
81  * in {Storage.data}.
82  * @function
83  * @private
84  * @param {String} namespace - the namespace of the app
85  * @param {Object} data      - the object that holds the app's settings
86  */
87 Storage.prototype._initialize = function(namespace, data) {
88   if (localStorage) {
89     if (!localStorage[namespace]) {
90       localStorage.setItem(namespace, JSON.stringify(data));
91     } else {
92       var store = localStorage.getItem(namespace);
93       return JSON.parse(store);
94     }
95   }
96
97   return data;
98 };
99
100 /**
101  * Stores the settings in the local storage.
```

```
102  * @function
103  * @private
104  * @param {Object} data - the data (settings) to be updated
105  */
106 Storage.prototype._save = function(data) {
107   if (localStorage && localStorage[this._db]) {
108     localStorage.setItem(this._db, JSON.stringify(data));
109   }
110   this.data = data;
111 };
112
113 module.exports = Storage;
```

Source file 10. src/js/app/template.js

```
1  /* jshint undef:true, unused:true, node:true, browser:true */
2  'use strict';
3
4  /**
5   * Creates an instance of a {Template},
6   * which appends node elements in the DOM or updates their inner content.
7   * @constructor
8   * @this {Template}
9   */
10 function Template() {
11 }
12
13 /**
14  * Returns the node element of the template
15  * for displaying warning messages.
16  * @function
17  * @param   {String} type    - the type of warning
18  * @param   {String} message - the text of the warning message
19  * @returns {Object}         - the top node element
20  */
21 Template.prototype.createAlertMessage = function(type, message) {
22   var div = document.createElement('div');
23
24   div.className = 'alert alert-warning';
25   div.innerHTML = [
26       '<table class="table table-condensed">',
27         '<tbody>',
28           '<tr>',
29             '<td class="h5"><i class="fa fa-exclamation-circle"></i></td>',
30             '<td>',
31               '<h5>', type, '</h5>',
32               '<p>', message, '</p>',
33             '</td>',
34           '</tr>',
35         '</tbody>',
36       '</table>'
37     ].join('');
38
39   return div;
40 };
41
42 /**
43  * Updates the inner HTML content of the output titles.
44  * @function
45  * @param {Array} texts - the array that holds information about the user input
46  */
47 Template.prototype.createOutputTitles = function(texts) {
48   var targets = [ document.getElementById('output-title-1'), document.
```

```
      getElementById('output-title-2') ],
49    tLength = targets.length;
50
51  for (var i = 0; i < tLength; i++) {
52    var fileName = texts[i].fileName || '';
53    var mode     = texts[i].inputMode;
54    var target   = targets[i];
55    target.innerHTML = [
56        '<p><b>', mode.toUpperCase(), ': </b>', fileName, '</p> ',
57      ].join('');
58  }
59 };
60
61 /**
62  * Returns the node element of the template
63  * for displaying the "PRINT OUTPUT" dialog.
64  * @function
65  * @param   {Array} texts - the array that holds information
66  *                          about the user input
67  * @returns {Object}      - the top node element
68  */
69 Template.prototype.createPrintDialog = function(texts) {
70   var section = document.createElement('section');
71
72   section.id = 'modal-print';
73   section.className = 'modal fade';
74   section.setAttribute('tabindex', '-1');
75   section.setAttribute('role', 'dialog');
76   section.innerHTML = [
77       '<div class="modal-dialog">',
78         '<div class="modal-content">',
79           '<div class="modal-header">',
80             '<button type="button" class="close" data-dismiss="modal" aria-label
                  ="Close">',
81               '<span aria-hidden="true">&times;</span>',
82             '</button>',
83             '<h4 class="modal-title">Print output</h4>',
84           '</div>',
85           '<div class="modal-body">',
86             '<div class="row">',
87               '<div class="col-xs-6">',
88                 '<div class="form-group form-group-sm">',
89                   '<label for="input-comment-1">1: Comment for ', texts[0].
                        inputMode, '</label>',
90                   '<textarea id="input-comment-1" class="form-control" rows="5"
                        autocomplete="off" placeholder="Type a comment"></textarea
                        >',
91                 '</div>',
92               '</div>',
93               '<div class="col-xs-6">',
94                 '<div class="form-group form-group-sm">',
95                   '<label for="input-comment-2">2: Comment for ', texts[1].
                        inputMode, '</label>',
96                   '<textarea id="input-comment-2" class="form-control" rows="5"
                        autocomplete="off" placeholder="Type a comment"></textarea
                        >',
97                 '</div>',
98               '</div>',
99             '</div>',
100          '</div>',
101          '<div class="modal-footer">',
102            '<button type="button" class="btn btn-default btn-sm" data-dismiss="
                   modal">Cancel</button>',
103            '<button id="modal-print-btn" type="button" class="btn btn-primary
```

```
                btn-sm">Print</button>',
104           '</div>',
105         '</div>',
106       '</div>'
107     ].join('');
108
109   return section;
110 };
111
112 /**
113  * Updates the inner HTML content of the hidden, on screen, node element
114  * that holds the information (statistics & comments) to be printed.
115  * @function
116  * @param {Array}  texts          - the array that holds information
117  *                                  about the user input
118  * @param {Number} uniqueMatches - the number of the unique matches found
119  */
120 Template.prototype.createPrintSummary = function(texts, uniqueMatches) {
121   var target = document.getElementById('print-summary');
122
123   target.innerHTML = [
124       '<h4>COMPARISON SUMMARY</h4>',
125       '<h6>DATE/TIME: ', (new Date()).toUTCString(), '</h6>',
126       '<table class="table table-condensed table-bordered">',
127         '<thead>',
128           '<tr>',
129             '<th class="col-xs-2"></th>',
130             '<th class="col-xs-5">', texts[0].fileName, '</th>',
131             '<th class="col-xs-5">', texts[1].fileName, '</th>',
132           '</tr>',
133         '</thead>',
134         '<tbody>',
135           '<tr>',
136             '<th>Comment</th>',
137             '<td id="print-comment-1"></td>',
138             '<td id="print-comment-2"></td>',
139           '</tr>',
140           '<tr>',
141             '<th>Type</th>',
142             '<td>', texts[0].inputMode, '</td>',
143             '<td>', texts[1].inputMode, '</td>',
144           '</tr>',
145           '<tr>',
146             '<th>Characters</th>',
147             '<td>', texts[0].nrOfCharacters, '</td>',
148             '<td>', texts[1].nrOfCharacters, '</td>',
149           '</tr>',
150           '<tr>',
151             '<th>Words</th>',
152             '<td>', texts[0].nrOfWords, '</td>',
153             '<td>', texts[1].nrOfWords, '</td>',
154           '</tr>',
155           '<tr>',
156             '<th>Unique matches</th>',
157             '<td colspan="2">', uniqueMatches, '</td>',
158           '</tr>',
159         '</tbody>',
160       '</table>'
161     ].join('');
162 };
163
164 /**
165  * Updates the inner HTML content
166  * of the node element that holds the statistical data.
```

```
167  * @function
168  * @param {Array}  texts         - the array that holds information
169  *                                 about the user input
170  * @param {Number} uniqueMatches - the number of the unique matches found
171  */
172 Template.prototype.createStatistics = function(texts, uniqueMatches) {
173   var target = document.getElementById('statistics');
174
175   target.innerHTML = [
176       '<table class="table table-condensed table-bordered">',
177         '<thead>',
178           '<tr>',
179             '<th class="col-xs-2"></th>',
180             '<th class="col-xs-5">', texts[0].fileName, '</th>',
181             '<th class="col-xs-5">', texts[1].fileName, '</th>',
182           '</tr>',
183         '</thead>',
184         '<tbody>',
185           '<tr>',
186             '<th>Type</th>',
187             '<td>', texts[0].inputMode, '</td>',
188             '<td>', texts[1].inputMode, '</td>',
189           '</tr>',
190           '<tr>',
191             '<th>Characters</th>',
192             '<td>', texts[0].nrOfCharacters, '</td>',
193             '<td>', texts[1].nrOfCharacters, '</td>',
194           '</tr>',
195           '<tr>',
196             '<th>Words</th>',
197             '<td>', texts[0].nrOfWords, '</td>',
198             '<td>', texts[1].nrOfWords, '</td>',
199           '</tr>',
200           '<tr>',
201             '<th>Unique matches</th>',
202             '<td colspan="2">', uniqueMatches, '</td>',
203           '</tr>',
204         '</tbody>',
205       '</table>'
206     ].join('');
207 };
208
209 module.exports = Template;
```

Source file 11. src/js/app/view.js

```
1  /* jshint undef:true, unused:true, node:true, browser:true */
2  'use strict';
3
4  var $           = require('jQuery');
5  var TargetMatch = require('../autoScroll/targetMatch.js');
6
7  /**
8   * Creates an instance of a {View},
9   * which implements all the UI logic of the application.
10  * @constructor
11  * @this  {View}
12  * @param {Template} template - the object that appends/updates elements
13  *                              in the DOM
14  */
15 function View(template) {
16   this.template = template;
17   this.results  = {};
```

```
18
19    // Selectors
20    this.$alertsPanel          = $('#alerts-panel');
21    this.$compareBtn           = $('#compare-btn');
22    this.$contentWrapper       = $('#content-wrapper');
23    this.$file                 = $(':file');
24    this.$htmlOptions          = $('#html-text-1, #html-text-2');
25    this.$inputLnk             = $('#input-lnk');
26    this.$inputPanel           = $('#input-panel');
27    this.$inputPanes           = $('#input-pane-1, #input-pane-2');
28    this.$inputFiles           = $('#input-file-1, #input-file-2');
29    this.$inputTexts           = $('#input-text-1, #input-text-2');
30    this.$outputPanel          = $('#output-panel');
31    this.$outputTexts          = $('#comparison-output-1, #comparison-output-2');
32    this.$outputTextContainers = $('#comparison-output-1 > .comparison-output-
          container, #comparison-output-2 > .comparison-output-container');
33    this.$outputParagraphs     = $('#comparison-output-1 > .comparison-output-
          container > p, #comparison-output-2 > .comparison-output-container > p');
34    this.$printBtn             = $('#print-btn');
35    this.$settingsSidebar      = $('#settings-sidebar');
36    this.$settingsSidebarLnk   = $('#settings-sidebar-lnk');
37    this.$settingsSidebarPanes = $('#comparison-options-pane, #input-options-pane'
          );
38    this.$spinner              = $('#min-match-length-spinner');
39    this.$tooltip              = $('[data-toggle="tooltip"], [rel="tooltip"]');
40
41    this._resetTextInputTabPanes();
42    this._updateOutputPanelHeight();
43    this._updateAlertsPanelWidth();
44  }
45
46  /**
47   * Binds events depending on the name specified.
48   * @function
49   * @param {String} event     - the name of the event
50   * @param {Function} handler - the callback function
51   */
52  View.prototype.bind = function(event, handler) {
53    var self = this;
54
55    switch (event) {
56      case 'changeSpinnerInput':
57        self.$spinner
58          .on('change mousewheel DOMMouseScroll', 'input[type="text"]', function(e
                ) {
59            var elem = e.target;
60            var id = self._getId(elem);
61              var minMatchLength = parseInt($(elem).val(), 10);
62
63              if (e.type === 'mousewheel' || e.type === 'DOMMouseScroll') {
64                // scrolling up
65                if (e.originalEvent.wheelDelta > 0 || e.originalEvent.detail <
                      0) {
66                    minMatchLength += 1;
67                }
68                // scrolling down
69                else {
70                    minMatchLength -= 1;
71                }
72              }
73
74              minMatchLength = (minMatchLength < 1) ? 1 : minMatchLength;
75
76            handler(id, minMatchLength);
```

```
77                self.updateUIOption(id, 'inputText', minMatchLength);
78              }
79            )
80            .on('click', '.btn', function(e) {
81              e.stopPropagation();
82
83              var $elem = $(e.delegateTarget).find('input[type="text"]');
84              var id = self._getId($elem);
85              var minMatchLength = parseInt($elem.val(), 10);
86
87              if ($(e.currentTarget).hasClass('plus')) {
88                minMatchLength += 1;
89              } else {
90                minMatchLength = (minMatchLength > 1) ? (minMatchLength - 1) :
                      minMatchLength;
91              }
92
93              handler(id, minMatchLength);
94                self.updateUIOption(id, 'inputText', minMatchLength);
95            });
96          break;
97
98        case 'compare':
99          self.$compareBtn.on('click', function(e) {
100           e.stopPropagation();
101
102           $(this).tooltip('hide');
103           self.$settingsSidebar.removeClass('expanded');
104           setTimeout(function() {
105             handler();
106           }, 200);
107         });
108         break;
109
110       case 'dismissAlert':
111         self.$alertsPanel.on('click', '.alert', function() {
112           $(this).remove();
113         });
114         break;
115
116       case 'initBootstrap':
117         self.$tooltip.tooltip({
118           container : 'body',
119           delay     : { "show": 800, "hide": 0 },
120           html      : true,
121           placement : 'bottom',
122           trigger   : 'hover'
123         });
124
125         self.$file.filestyle({
126           buttonName  : "btn-primary",
127           buttonText  : "Browse file",
128           placeholder : "No file selected",
129           size        : "sm"
130         });
131         break;
132
133       case 'inputFile':
134         self.$inputFiles.on('change', function(e) {
135           var elem = e.target;
136           var id = self._getId(elem);
137
138           var tabPaneId = self._getId($(elem).parents('.tab-pane'));
139           self.toggleErrorStatus('hide', tabPaneId);
```

```
140
141         var file = elem.files[0];
142         var idx = self._getIndex(id);
143         var loadingElem = $(elem).parent();
144         handler(file, idx, loadingElem, tabPaneId);
145       });
146       break;
147
148     case 'inputText':
149       self.$inputTexts.on('change input', function(e) {
150         var elem = e.target;
151         var $elem = $(elem);
152         var tabPaneId = self._getId($elem.parents('.tab-pane'));
153
154         if (e.type === 'input') {
155           self.toggleErrorStatus('hide', tabPaneId);
156         }
157
158         if (e.type === 'change') {
159           var id = self._getId(elem);
160           var text = $elem.val();
161           var idx = self._getIndex(id);
162           handler(text, idx, tabPaneId);
163         }
164       });
165       break;
166
167     case 'hidePrintDialog':
168       self.$contentWrapper.on('hide.bs.modal', '.modal', function(e) {
169         self._togglePrintDialog('hide', e.target);
170       });
171       break;
172
173     case 'print':
174       self.$contentWrapper.on('click', '#modal-print-btn', function(e) {
175         e.stopPropagation();
176
177         var inputComment1  = $('#input-comment-1').val();
178         var inputComment2  = $('#input-comment-2').val();
179         $('#print-comment-1').text(inputComment1);
180         $('#print-comment-2').text(inputComment2);
181
182         var hideModalPromise = $('.modal').modal('hide').promise();
183         handler(hideModalPromise);
184       });
185       break;
186
187     case 'resize':
188       $(window).on('resize', function() {
189         self._updateOutputPanelHeight();
190         self._updateAlertsPanelWidth();
191       });
192       break;
193
194     case 'scrollToMatch':
195       self.$outputTexts.on('click', 'a', function(e) {
196         e.preventDefault();
197         e.stopPropagation();
198
199         var targetMatch = new TargetMatch(e.target);
200         var scrollPosition = targetMatch.getScrollPosition();
201         targetMatch.scroll(scrollPosition);
202       });
203       break;
```

```
204
205       case 'selectHTMLOption':
206         self.$inputPanel.on('change', 'input[type="checkbox"]', function(e) {
207           var elem = e.target;
208           var id = self._getId(elem);
209           var idx = self._getIndex(id);
210           var newValue = $(elem).prop('checked');
211           var text = self.$inputTexts.eq(idx).val();
212           handler(idx, newValue, text);
213         });
214         break;
215
216       case 'selectSettingsOption':
217         self.$settingsSidebarPanes.on('change', 'input[type="checkbox"]', function
              (e) {
218           var elem = e.target;
219           var id = self._getId(elem);
220           var newValue = $(elem).prop('checked');
221           handler(id, newValue);
222         });
223         break;
224
225       case 'selectTab':
226         self.$inputPanes.on('shown.bs.tab', 'a[data-toggle="tab"]', function(e) {
227           var lastTabPaneId = $(e.relatedTarget).attr('href');
228           self.toggleErrorStatus('hide', lastTabPaneId);
229           });
230         break;
231
232       case 'showPrintDialog':
233         self.$printBtn.on('click', function(e) {
234           e.stopPropagation();
235           self._togglePrintDialog('show');
236         });
237         break;
238
239       case 'toggleInputPanel':
240         self.$inputLnk.on('click', function(e) {
241           e.preventDefault();
242           e.stopPropagation();
243           // Hide tooltip (if any)
244           $(this).tooltip('hide');
245           self._toggleInputPanel('toggle');
246         });
247         break;
248
249       case 'toggleSettingsSidebar':
250         self.$settingsSidebarLnk.on('click', function(e) {
251           e.preventDefault();
252           e.stopPropagation();
253           // Hide tooltip (if any)
254           $(this).tooltip('hide');
255           self.$settingsSidebar.toggleClass('expanded');
256         });
257
258         // Hide settings sidebar when clicking inside the 'nav' and '#content-
              wrapper' elements
259         $('body').on('click', 'nav, #content-wrapper', function() {
260           self.$settingsSidebar.removeClass('expanded');
261         });
262         break;
263
264       case 'toggleSettingsSidebarPanes':
265         self.$settingsSidebar.on('click', '.panel-title', function() {
```

```
266         $(this).toggleClass('active');
267       });
268       break;
269
270     default:
271       throw new Error('Event type not valid.');
272   }
273 };
274
275 /**
276  * Removes all <p> nodes from each output pane
277  * and hides the output panel.
278  * @function
279  */
280 View.prototype.clearOutputPanel = function() {
281   var self = this;
282
283   self.$outputParagraphs.each(function() {
284     $(this).remove();
285   });
286   self._toggleOutputPanel('hide');
287   self.toggleWaitingCursor('hide');
288 };
289
290 /**
291  * Clears all input from the "FILE" tab pane.
292  * @function
293  * @param {Number} idx - the number of the tab pane
294  *                       0: for left-side pane, 1: for right-side pane
295  */
296 View.prototype.clearTabPaneFileInput = function(idx) {
297   var self = this;
298   var tabPaneId = '#tab-file-' + (idx + 1);
299   $(tabPaneId + ' input').filestyle('clear');
300   self.toggleErrorStatus('hide', tabPaneId);
301   self.loading('cancel', tabPaneId);
302 };
303
304 /**
305  * Clears all input from the "TEXT" tab pane.
306  * @function
307  * @param {Number} idx - the number of the tab pane
308  *                       0: for left-side pane, 1: for right-side pane
309  */
310 View.prototype.clearTabPaneTextInput = function(idx) {
311   var self = this;
312   var tabPaneId = '#tab-text-' + (idx + 1);
313   $(tabPaneId + ' textarea').val('');
314   self.toggleErrorStatus('hide', tabPaneId);
315 };
316
317 /**
318  * Creates the node templates.
319  * @function
320  */
321 View.prototype.createTemplates = function() {
322   var self = this;
323   self.template.createPrintSummary(self.results.texts, self.results.
        uniqueMatches);
324   self.template.createStatistics(self.results.texts, self.results.uniqueMatches)
        ;
325   self.template.createOutputTitles(self.results.texts);
326 };
327
```

```
328 /**
329  * Returns the ids of active tab panes as an array of strings.
330  * @function
331  * @returns {Array<String>} - the ids of the active tab panes
332  */
333 View.prototype.getActiveTabPaneIds = function() {
334   var self = this,
335     tabPaneIds = [];
336
337   $('.tab-pane.active').each(function() {
338     var tabPaneId = self._getId(this);
339     tabPaneIds.push(tabPaneId);
340   });
341   return tabPaneIds;
342 };
343
344 /**
345  * Shows/hides an node element depending on the event specified.
346  * Used to show the progress of a process (e.g. input reading).
347  * @function
348  * @param {String} event  - the name of the event
349  * @param {Object} target - the id of the node element
350  */
351 View.prototype.loading = function(event, target) {
352   var self = this;
353
354   switch (event) {
355     case 'start':
356       self.toggleCompareBtn('disable');
357       $(target).find('.fa').addClass('hidden');
358       $(target).find('.fa-spinner').removeClass('hidden');
359       break;
360
361     case 'done':
362       self.toggleCompareBtn('enable');
363       $(target).find('.fa').addClass('hidden');
364       $(target).find('.fa-check').removeClass('hidden');
365       break;
366
367     case 'cancel':
368       self.toggleCompareBtn('enable');
369       $(target).find('.fa').addClass('hidden');
370       break;
371
372     case 'error':
373       self.toggleCompareBtn('enable');
374       $(target).find('.fa').addClass('hidden');
375       $(target).find('.fa-times').removeClass('hidden');
376       break;
377
378     default:
379       throw new Error('Event type not valid.');
380   }
381 };
382
383 /**
384  * Resets the scroll bars.
385  * @function
386  */
387 View.prototype.resetScrollbars = function() {
388   var self = this;
389   self.$outputTexts.scrollTop(0);
390 };
391
```

```
392 /**
393  * Clears text from textarea and unchecks checkboxes.
394  * Important for Internet Explorer,
395  * since it does not recognize the "autocomplete='off'" attribute.
396  * @function
397  * @private
398  */
399 View.prototype._resetTextInputTabPanes = function() {
400   var self = this;
401   self.$htmlOptions.prop('checked', false);
402   self.$inputTexts.val('');
403 };
404
405 /**
406  * Displays a warning message.
407  * @function
408  * @param {String} type    - the type of the message
409  * @param {String} message - the text of the message
410  * @param {Number} delay   - the time in milliseconds, during which the message
411  *                           should remain visible
412  */
413 View.prototype.showAlertMessage = function(type, message, delay) {
414   var self = this,
415       alertMessage = self.template.createAlertMessage(type, message);
416
417   self.$alertsPanel.append($(alertMessage));
418   setTimeout(function() {
419     self.$alertsPanel.children().eq(0).remove();
420   }, delay);
421 };
422
423 /**
424  * Appends the array of nodes returned by the comparison
425  * to the <p> node element of each output pane
426  * and shows the output panel.
427  * @function
428  * @param {Array} nodes - the array of nodes returned by the comparison
429  */
430 View.prototype.showSimilarities = function(nodes) {
431   var self = this,
432       nLength = nodes.length;
433
434   for (var i = 0; i < nLength; i++) {
435     var $p = $('<p>').append(nodes[i]);
436     self.$outputTextContainers.eq(i).html($p);
437   }
438
439   self._toggleOutputPanel('show');
440   setTimeout(function() {
441     self._toggleInputPanel('hide');
442   }, 100);
443
444   self.toggleWaitingCursor('hide');
445 };
446
447 /**
448  * Enables/disables the compare button
449  * depending on the event specified.
450  * @function
451  * @param {String} event - the name of the event
452  */
453 View.prototype.toggleCompareBtn = function(event) {
454   var self = this;
455   switch (event) {
```

```
456       case 'enable':
457         self.$compareBtn.prop('disabled', false);
458         break;
459
460       case 'disable':
461         self.$compareBtn.prop('disabled', true);
462         break;
463
464       default:
465         throw new Error('Event type not valid.');
466     }
467 };
468
469 /**
470  * Toggles the class "has-error",
471  * which applies a red border around input node elements,
472  * to prompt the user in case of erroneous input.
473  * @function
474  * @param {String} event     - the name of the event
475  * @param {String} tabPaneId - the id of the tab pane
476  */
477 View.prototype.toggleErrorStatus = function(event, tabPaneId) {
478   switch (event) {
479     case 'show':
480       $(tabPaneId + ' .apply-error').addClass('has-error');
481       break;
482
483     case 'hide':
484       $(tabPaneId + ' .apply-error').removeClass('has-error');
485       break;
486
487     default:
488       throw new Error('Event type not valid.');
489   }
490 };
491
492 /**
493  * Toggles the style of the cursor (from "default" to "waiting", and vice versa)
494  * depending on the event specified.
495  * @function
496  * @param {String} event - the name of the event
497  */
498 View.prototype.toggleWaitingCursor = function(event) {
499   switch (event) {
500     case 'show':
501       document.body.className = 'waiting';
502       break;
503
504     case 'hide':
505       document.body.className = '';
506       break;
507
508     default:
509       throw new Error('Event type not valid.');
510   }
511 };
512
513 /**
514  * Updates the value of a setting in the UI.
515  * @function
516  * @param {String}           id    - the id of the control element
517  * @param {String}           type  - the type of the control element
518  * @param {(Boolean|Number)} value - the value of the setting
519  */
```

```
520 View.prototype.updateUIOption = function(id, type, value) {
521   switch (type) {
522     case 'checkbox':
523       $(id).prop('checked', value);
524       break;
525     case 'select':
526       $(id).val(value);
527       break;
528     default:
529       $(id).val(value);
530   }
531 };
532
533 /**
534  * Calculates the height of the output pane
535  * so that it fits entirely in the window.
536  * @function
537  * @private
538  */
539 View.prototype._computeOutputPanelHeight = function() {
540   var self = this;
541   var bodyHeight = $('body').outerHeight(true);
542   var outputPos  = self.$outputPanel.offset().top;
543   var outputTopPadding = parseInt(self.$outputPanel.css('padding-top'), 10);
544   var elemPos    = self.$outputTexts.eq(0).offset().top;
545   var posOffset  = (elemPos - outputPos);
546   return bodyHeight - outputPos - (posOffset + outputTopPadding);
547 };
548
549 /**
550  * Returns the id of a node element as a string (e.g. "#id").
551  * @function
552  * @param   {Object} target - the id of the node element
553  * @returns {String}        - the string of the node element's id
554  */
555 View.prototype._getId = function(target) {
556   return '#' + $(target).attr('id');
557 };
558
559 /**
560  * Returns the number contained in the id of a node element.
561  * @function
562  * @private
563  * @param   {String} id - the id of the node element
564  * @returns {Number}    - the number of the id
565  */
566 View.prototype._getIndex = function(id) {
567   var tokens = id.split('-');
568   var idx = tokens[tokens.length - 1];
569   return parseInt(idx, 10) - 1;
570 };
571
572 View.prototype._toggleInputPanel = function(event) {
573   var self = this;
574   switch (event) {
575     case 'toggle':
576       $('.btn-group.open').removeClass('open');
577       self.$inputPanel.toggleClass('expanded');
578       break;
579
580     case 'hide':
581       self.$inputPanel.removeClass('expanded');
582       break;
583
```

```
584       default:
585         throw new Error('Event type not valid.');
586     }
587   };
588
589   /**
590    * Shows/hides the output panel depending on the event specified.
591    * @function
592    * @private
593    * @param {String} event - the name of the event
594    */
595   View.prototype._toggleOutputPanel = function(event) {
596     var self = this;
597     switch (event) {
598       case 'show':
599         self.$outputPanel.removeClass('invisible');
600         break;
601
602       case 'hide':
603         self.$outputPanel.addClass('invisible');
604         break;
605
606       default:
607         throw new Error('Event type not valid.');
608     }
609   };
610
611   /**
612    * Shows/hides the "PRINT OUTPUT" dialog depending on the event specified.
613    * @function
614    * @private
615    * @param {String} event  - the name of the event
616    * @param {Object} target - the node element to be removed
617    */
618   View.prototype._togglePrintDialog = function(event, target) {
619     var self = this;
620     switch (event) {
621       case 'show':
622         var $printDialog = $(self.template.createPrintDialog(self.results.texts));
623         self.$contentWrapper.append($printDialog);
624         $printDialog.modal('show');
625         break;
626
627       case 'hide':
628         $(target).remove();
629         break;
630
631       default:
632         throw new Error('Event type not valid.');
633     }
634   };
635
636   /**
637    * Updates the width of the alerts' panel.
638    * @function
639    * @private
640    */
641   View.prototype._updateAlertsPanelWidth = function() {
642     var self        = this,
643         marginLR    = 3 * 2,
644         navWidth    = $('nav').width(),
645         navLeftWidth  = $('nav .pull-left').outerWidth(),
646         navRightWidth = $('nav .pull-right').outerWidth(),
647         maxWidth    = navWidth - (navLeftWidth + navRightWidth + marginLR);
```

```
648
649   self.$alertsPanel.css({
650     'left'      : navLeftWidth + 'px',
651     'max-width' : maxWidth + 'px'
652   });
653 };
654
655 /**
656  * Updates the height of each output pane.
657  * @function
658  * @private
659  */
660 View.prototype._updateOutputPanelHeight = function() {
661   var self = this,
662       h = self._computeOutputPanelHeight();
663
664   self.$outputTexts.each(function() {
665     $(this).css('height', h + 'px');
666   });
667 };
668
669 module.exports = View;
```

Source file 12. src/js/autoScroll/scrollPosition.js

```
1  /* jshint undef:true, unused:true, node:true, browser:true */
2  'use strict';
3
4  /**
5   * Creates an instance of a {ScrollPosition}.
6   * @constructor
7   * @this  {ScrollPosition}
8   * @param {Number} topPadding    - the top padding
9   * @param {Number} bottomPadding - the bottom padding
10  * @param {Number} yPosition     - the vertical position of the scroll bar
11  */
12 function ScrollPosition(topPadding, bottomPadding, yPosition) {
13   this.topPadding    = topPadding;
14   this.bottomPadding = bottomPadding;
15   this.yPosition     = yPosition;
16 }
17
18 module.exports = ScrollPosition;
```

Source file 13. src/js/autoScroll/targetMatch.js

```
1  /* jshint undef:true, unused:true, node:true, browser:true */
2  'use strict';
3
4  var $              = require('jQuery');
5  var ScrollPosition = require('./scrollPosition.js');
6
7  /**
8   * Creates an instance of a {TargetMatch},
9   * which hold information on the target match node element.
10  * @constructor
11  * @this  {TargetMatch}
12  * @param {elem} elem - the source match node
13  */
14 function TargetMatch(elem) {
15   this.$srcElem            = $(elem);
16   this.$srcParent          = $(this.$srcElem.parent().parent().parent());
```

```
17
18    this.$elem                = $(this.$srcElem.attr('href'));
19    this.$wrapper             = $(this.$elem.parent());
20    this.$container           = $(this.$wrapper.parent());
21    this.$parent              = $(this.$container.parent());
22
23    this.parentHeight         = this.$parent[0].getBoundingClientRect().height;
24    this.containerTBPadding   = parseInt(this.$container.css('padding-top'), 10) +
          parseInt(this.$container.css('padding-bottom'), 10);
25    this.wrapperTopPadding    = parseFloat(this.$wrapper.css('padding-top'));
26    this.wrapperBottomPadding = parseFloat(this.$wrapper.css('padding-bottom'));
27  }
28
29  /**
30   * Returns the new scroll position of the target match node.
31   * @function
32   * @returns {ScrollPosition} - the new scroll position
33   */
34  TargetMatch.prototype.getScrollPosition = function() {
35    var self                = this,
36        wrapperBottom       = self.$wrapper.outerHeight(true) + self.
              containerTBPadding,
37        wrapperTopPadding   = self.wrapperTopPadding,
38        wrapperBottomPadding = self.wrapperBottomPadding,
39        // Calculate difference on the y axis (relative to parent element)
40        yPosDiff            = (self.$srcElem.offset().top - self.$srcParent.
              offset().top) - (self.$elem.offset().top - self.$parent.offset().top);
41
42    // Remove top padding
43    if (wrapperTopPadding > 0) {
44      yPosDiff += wrapperTopPadding;
45      wrapperBottom -= wrapperTopPadding;
46      wrapperTopPadding = 0;
47    }
48
49    // Remove bottom padding
50    if (wrapperBottomPadding > 0) {
51      wrapperBottom -= wrapperBottomPadding;
52      wrapperBottomPadding = 0;
53    }
54
55    // Compute new scroll position
56    var yScrollPos = self.$parent.scrollTop() - yPosDiff;
57
58    // Add bottom padding, if needed
59    if (yScrollPos > (wrapperBottom - self.parentHeight)) {
60      var bottomOffset = (yScrollPos + self.parentHeight) - (wrapperBottom);
61      wrapperBottomPadding = Math.abs(bottomOffset);
62    }
63
64    // Add top padding, if needed
65    if (yScrollPos < 0) {
66      var topOffset = yScrollPos;
67      wrapperTopPadding = Math.abs(topOffset);
68      yScrollPos -= topOffset;
69    }
70
71    return new ScrollPosition(wrapperTopPadding, wrapperBottomPadding, yScrollPos)
          ;
72  };
73
74  /**
75   * Animates scrolling to the new position.
76   * @function
```

```
 * @param {ScrollPosition} scrollPosition - the new scroll position
 */
TargetMatch.prototype.scroll = function(scrollPosition) {
  var self = this;

  self.$wrapper.animate({
    'padding-top'    : scrollPosition.topPadding,
    'padding-bottom' : scrollPosition.bottomPadding,
  }, 700);

  self.$parent.animate({
    'scrollTop'      : scrollPosition.yPosition,
  }, 700);

};

module.exports = TargetMatch;
```

Source file 14. src/js/inputReader/fileInputReader.js

```
/* jshint undef:true, unused:true, node:true, browser:true */
'use strict';

var $     = require('jQuery');
var JSZip = require('JSZip');

/**
 * Creates an instance of a {FileInputReader},
 * which parses and extracts the text contents of the DOCX, ODT and TXT files.
 * @constructor
 * @this  {FileInputReader}
 * @param {File}    file            - the file selected by the user
 * @param {Boolean} ignoreFootnotes - the option for including/excluding
 *                                    the document's footnotes from parsing
 */
function FileInputReader(file, ignoreFootnotes) {
  this.file            = file;
  this.ignoreFootnotes = ignoreFootnotes;
}

/**
 * Returns a promise that handles the file reading.
 * When resolved, the contents of the file are returned as a string.
 * @function
 * @param   {Function} loadingStarted - the callback function
 *                                      for the onloadstart event
 * @returns {Promise}
 */
FileInputReader.prototype.readFileInput = function(loadingStarted) {
  var self     = this,
      file     = self.file,
      fileType = self._getFileType(),
      deferred = $.Deferred(),
      fr       = new FileReader();

  fr.onerror = function(e) {
    var error = e.target.error;
    switch (error.code) {
      case error.NOT_FOUND_ERR:
        deferred.reject('File not found!');
        break;
      case error.NOT_READABLE_ERR:
        deferred.reject('File not readable.');
```

```
        break;
      case error.ABORT_ERR:
        deferred.reject('File reading aborted.');
        break;
      default:
        deferred.reject('An error occurred while reading this file.');
    }
  };

  fr.onloadstart = loadingStarted;

  switch (fileType) {
    case 'docx':
      fr.onload = function(e) {
        var docxText = self._readDOCX(e.target.result);

        if (docxText) {
          if (/\S/.test(docxText)) {
            deferred.resolve(docxText);
          } else {
            deferred.reject('The selected DOCX file is empty.');
          }
        } else {
          deferred.reject('The selected file is not a valid DOCX file.');
        }
      };
      fr.readAsArrayBuffer(file);
      break;

    case 'odt':
      fr.onload = function(e) {
        var odtText = self._readODT(e.target.result);

        if (odtText) {
          if (/\S/.test(odtText)) {
            deferred.resolve(odtText);
          } else {
            deferred.reject('The selected ODT file is empty.');
          }
        } else {
          deferred.reject('The selected file is not a valid ODT file.');
        }
      };
      fr.readAsArrayBuffer(file);
      break;

    case 'txt':
      fr.onload = function(e) {
        var txtText = e.target.result;

        if (txtText) {
          if (/\S/.test(txtText)) {
            // Mac uses carriage return, which is not processed correctly
            // Replace each carriage return, not followed by a line feed
            // with a line feed
            var crCleanedText = txtText.replace(/\r(?!\n)/g, '\n');
            deferred.resolve(crCleanedText);
          } else {
            deferred.reject('The selected TXT file is empty.');
          }
        }
      };
      fr.readAsText(file);
      break;
```

```
108
109       default:
110         deferred.reject('File type not supported.');
111     }
112
113     return deferred.promise();
114   };
115
116   /**
117    * Traverses recursively all children starting from the top XML node,
118    * irrespective of how deep the nesting is.
119    * Returns their text contents as a string.
120    * @function
121    * @private
122    * @param   {Object} node       - the top XML node element
123    * @param   {String} tSelector  - the selector for text elements
124    * @param   {String} brSelector - the selector for soft line breaks
125    * @returns {String}            - the text content of the node
126    */
127   FileInputReader.prototype._extractTextFromNode = function(node, tSelector,
       brSelector) {
128     var self = this,
129         // Paragraph selectors for both DOCX and ODT,
130         // supported both by Chrome and other browsers
131         // Chrome uses different selectors
132         delimeters = {
133           'w:p'    : '\n',
134           'text:p' : '\n',
135           'p'      : '\n'
136         },
137         delimeter = delimeters[node.nodeName] || '',
138         str  = '';
139
140     if (node.hasChildNodes()) {
141       var child = node.firstChild;
142
143       while (child) {
144         // These selectors apply only to the footnotes of ODT files
145         // Footnotes should appear all together at the end of the extracted text
146         // and not inside the text at the point where the reference is.
147         if (child.nodeName === 'text:note' || child.nodeName === 'note') {
148           child = child.nextSibling;
149           continue;
150         }
151
152         if (child.nodeName === tSelector) {
153           str += child.textContent;
154         } else if (child.nodeName === brSelector) {
155           str += '\n';
156         }
157         else {
158           str += self._extractTextFromNode(child, tSelector, brSelector);
159         }
160
161         child = child.nextSibling;
162       }
163     }
164
165     return str + delimeter;
166   };
167
168   /**
169    * Returns the type of file depending on the file's extension.
170    * @function
```

```
171  * @private
172  * @param   {Object} file - the file selected by the user
173  * @returns {String}      - the type of file
174  */
175 FileInputReader.prototype._getFileType = function() {
176   var self = this,
177       file = self.file;
178
179   if (/docx$/i.test(file.name)) {
180     return 'docx';
181   }
182
183   if (/odt$/i.test(file.name)) {
184     return 'odt';
185   }
186
187   if (/txt$/i.test(file.name)) {
188     return 'txt';
189   }
190
191   return undefined;
192 };
193
194 /**
195  * Returns the contents of all XML nodes as a string.
196  *
197  * @function
198  * @private
199  * @param   {Object[]} nodes    - the array of XML nodes
200  * @param   {String} tSelector  - the selector for text elements
201  * @param   {String} brSelector - the selector for soft line breaks
202  * @returns {String}            - the text content of all XML nodes
203  */
204 FileInputReader.prototype._getTextContent = function(nodes, tSelector,
      brSelector) {
205   var self    = this,
206       nLength = nodes.length,
207       textContent;
208
209   for (var i = 0; i < nLength; i++) {
210     var node = nodes[i];
211     var nodeContent = self._extractTextFromNode(node, tSelector, brSelector);
212     textContent = [textContent, nodeContent].join('');
213   }
214
215   return textContent;
216 };
217
218 /**
219  * Returns the contents of the DOCX file as a string.
220  * @function
221  * @private
222  * @param   {Object} fileContents - the contents of the file object
223  * @returns {String}              - the text of the DOCX file
224  */
225 FileInputReader.prototype._readDOCX = function(fileContents) {
226   var self = this,
227       document,
228       footnotes  = '',
229       xmlDoc,
230       tSelector  = 'w:t',
231       brSelector = 'w:br',
232       zip;
233
```

```
234   // Unzip the file
235   try {
236     zip = new JSZip(fileContents);
237
238     // Read the main text of the DOCX file
239     var file = zip.files['word/document.xml'];
240
241     if (file) {
242       xmlDoc = $.parseXML(file.asText());
243       var pNodes = $(xmlDoc).find('w\\:body, body').children();
244       document = self._getTextContent(pNodes, tSelector, brSelector);
245     }
246
247     // Read footnotes/endnotes
248     if (!self.ignoreFootnotes) {
249       // Read footnotes
250       file = zip.files['word/footnotes.xml'];
251       if (file) {
252         xmlDoc = $.parseXML(file.asText());
253         var fNodes = $(xmlDoc).find('w\\:footnotes, footnotes').children('w\\:
              footnote:not([w\\:type]), footnote:not([type])');
254         var fNodesText = self._getTextContent(fNodes, tSelector, brSelector);
255         if (fNodesText) {
256           footnotes = [footnotes, fNodesText].join('');
257         }
258       }
259
260       // Read endnotes
261       file = zip.files['word/endnotes.xml'];
262       if (file) {
263         xmlDoc = $.parseXML(file.asText());
264         var eNodes = $(xmlDoc).find('w\\:endnotes, endnotes').children('w\\:
              endnote:not([w\\:type]), endnote:not([type])');
265         var eNodesText = self._getTextContent(eNodes, tSelector, brSelector);
266         if (eNodesText) {
267           footnotes = [footnotes, eNodesText].join('');
268         }
269       }
270
271       if (footnotes && footnotes.length) {
272         document = [document, 'FOOTNOTES', footnotes].join('\n');
273       }
274     }
275   } catch (error) {
276
277   }
278
279   return document;
280 };
281
282 /**
283  * Returns the contents of the ODT file as a string.
284  * @function
285  * @private
286  * @param   {Object} fileContents - the contents of the file object
287  * @returns {String}              - the text of the ODT file
288  */
289 FileInputReader.prototype._readODT = function(fileContents) {
290   var self = this,
291       document,
292       tSelector  = '#text',
293       brSelector = 'text:line-break',
294       zip;
295
```

```
296   // Unzip the file
297   try {
298     zip = new JSZip(fileContents);
299
300     // Read the main text, as well as the footnotes/endnotes of the ODT file
301     var file = zip.files['content.xml'];
302
303     if (file) {
304       var xmlDoc = $.parseXML(file.asText());
305       var pNodes = $(xmlDoc).find('office\\:body, body').children();
306       document = self._getTextContent(pNodes, tSelector, brSelector);
307
308       if (!self.ignoreFootnotes) {
309         var fNodes = $(pNodes).find('text\\:note-body, note-body');
310         var footnotes = self._getTextContent(fNodes, tSelector, brSelector);
311
312         if (footnotes && footnotes.length) {
313           document = [document, 'FOOTNOTES', footnotes].join('\n');
314         }
315       }
316     }
317   } catch (error) {
318
319   }
320
321   return document;
322 };
323
324 module.exports = FileInputReader;
```

Source file 15. src/js/inputReader/textInputReader.js

```
1  /* jshint undef:true, unused:true, node:true, browser:true */
2  'use strict';
3
4  var $       = require('jQuery');
5  var XRegExp = require('XRegExp');
6
7  /**
8   * Creates an instance of a {TextInputReader},
9   * which parses and extracts the text contents of the HTML text input.
10  * @constructor
11  * @this {TextInputReader}
12  */
13 function TextInputReader() {
14 }
15
16 /**
17  * Returns a promise that handles the HTML input reading.
18  * When resolved, the contents of the HTML text
19  * are returned as a string.
20  * @function
21  * @param   {String} text - the HTML text input
22  * @returns {Promise}
23  */
24 TextInputReader.prototype.readTextInput = function(text) {
25   var self     = this,
26       deferred = $.Deferred();
27
28   var cleanedText = '';
29   var div = document.createElement('div');
30   div.innerHTML = text;
31
```

```
32    var textNode = self._extractTextFromNode(div);
33    // If is not empty or not contains only white spaces
34    if (textNode.length && /\S/.test(textNode)) {
35      cleanedText = [cleanedText, textNode].join('');
36      // Remove multiple white spaces
37      cleanedText = cleanedText.replace(/\n[ \t\v]*/g, '\n');
38      // Remove multiple newlines
39      cleanedText = cleanedText.replace(/\n{3,}/g, '\n\n');
40
41      // Resolve
42      deferred.resolve(cleanedText);
43    } else {
44      // Reject
45      deferred.reject('HTML input has no valid text contents.');
46    }
47
48    return deferred.promise();
49  };
50
51  /**
52   * Traverses recursively all child nodes,
53   * irrespective of how deep the nesting is.
54   * Returns the HTML text contents as a string.
55   * @function
56   * @private
57   * @param   {Object} node - the parent HTML node element
58   * @returns {String}      - the text content of the HTML string
59   */
60  TextInputReader.prototype._extractTextFromNode = function(node) {
61    var self = this,
62        // Match any letter
63        letterRegex = XRegExp('^\\pL+$'),
64        str = '';
65
66    // Returns whether a node should be skipped
67    var isValidNode = function(nodeName) {
68      var skipNodes       = ['IFRAME', 'NOSCRIPT', 'SCRIPT', 'STYLE'],
69          skipNodesLength = skipNodes.length;
70
71      for (var i = 0; i < skipNodesLength; i++) {
72        if (nodeName === skipNodes[i]) {
73          return false;
74        }
75      }
76      return true;
77    };
78
79    if (isValidNode(node.nodeName) && node.hasChildNodes()) {
80      var child = node.firstChild;
81
82      while (child) {
83        // If text node
84        if (child.nodeType === 3) {
85          var content = child.textContent;
86          if (content.length) {
87            str += content;
88          }
89        } else {
90          var extractedContent = self._extractTextFromNode(child);
91          // Add a space between text nodes that are not separated
92          // by a space or newline (e.g. as in lists)
93          if (letterRegex.test(str[str.length - 1]) && letterRegex.test(
                extractedContent[0])) {
94            str += ' ';
```

```
95          }
96          str += extractedContent;
97        }
98
99        child = child.nextSibling;
100     }
101   }
102
103   return str;
104 };
105
106 module.exports = TextInputReader;
```

Source file 16. src/js/simtexter/match.js

```
1  /* jshint undef:true, unused:true, node:true, browser:true */
2  'use strict';
3
4  /**
5   * Records a match found in the source and the target text.
6   * @constructor
7   * @this  {Match}
8   * @param {Number} srcTxtIdx     - the index of the source text
9   *                                 in {SimTexter.texts[]}, where the match
10  *                                 is found
11  * @param {Number} srcTkBeginPos - the index of the source text's token
12  *                                 in {SimTexter.tokens[]}, where the match
13  *                                 starts
14  * @param {Number} trgTxtIdx     - the index of the target text
15  *                                 in {SimTexter.texts[]}, where the match
16  *                                 is found
17  * @param {Number} trgTkBeginPos - the index of the target text's token
18  *                                 in {SimTexter.tokens[]}, where the match
19  *                                 starts
20  * @param {Number} matchLength   - the length of the match
21  */
22 function Match(srcTxtIdx, srcTkBeginPos, trgTxtIdx, trgTkBeginPos, matchLength)
        {
23   this.srcTxtIdx     = srcTxtIdx;
24   this.srcTkBeginPos = srcTkBeginPos;
25   this.trgTxtIdx     = trgTxtIdx;
26   this.trgTkBeginPos = trgTkBeginPos;
27   this.matchLength   = matchLength;
28 }
29
30 module.exports = Match;
```

Source file 17. src/js/simtexter/matchSegment.js

```
1  /* jshint undef:true, unused:true, node:true, browser:true */
2  'use strict';
3
4  /**
5   * Records a match found in a text.
6   * @constructor
7   * @this  {MatchSegment}
8   * @param {Number} txtIdx      - the index of the text in {SimTexter.texts[]},
9   *                               where the match has been found
10  * @param {Number} tkBeginPos  - the index of the token in {SimTexter.tokens[]},
11  *                               where the match starts
12  * @param {Number} matchLength - the length of the match
13  */
```

```
14 function MatchSegment(txtIdx, tkBeginPos, matchLength) {
15   this.txtIdx      = txtIdx;
16   this.tkBeginPos  = tkBeginPos;
17   this.matchLength = matchLength;
18   this.styleClass  = undefined;
19 }
20
21 /**
22  * Returns the match's link node.
23  * @function
24  * @param {String}       text            - the text content of the node
25  * @param {MatchSegment} trgMatchSegment - the target match segment
26  * @returns                              - the match's link node
27  */
28 MatchSegment.prototype.createLinkNode = function(text, trgMatchSegment) {
29   var self = this,
30       matchLink = document.createElement('a');
31
32     matchLink.id          = [self.txtIdx + 1, '-', self.tkBeginPos].join('');
33     matchLink.className   = self.styleClass;
34     matchLink.href        = ['#', trgMatchSegment.txtIdx+1, '-', trgMatchSegment
           .tkBeginPos].join('');
35     matchLink.textContent = text;
36     return matchLink;
37 };
38
39 /**
40  * Returns the index of the token in {SimTexter.tokens[]},
41  * where the match ends.
42  * @function
43  * @returns {Number} - the last token position of the match (non-inclusive)
44  */
45 MatchSegment.prototype.getTkEndPosition = function() {
46   var self = this;
47   return self.tkBeginPos + self.matchLength;
48 };
49
50 /**
51  * Returns the index of the character in the input string,
52  * where the match starts.
53  * @function
54  * @returns {Number} - the first character of the match in the input string
55  */
56 MatchSegment.prototype.getTxtBeginPos = function(tokens) {
57   var self = this;
58     return tokens[self.tkBeginPos].txtBeginPos;
59 };
60
61 /**
62  * Returns the index of the character in the input string,
63  * where the match ends.
64  * @function
65  * @returns {Number} - the last character of the match in the input string
66  */
67 MatchSegment.prototype.getTxtEndPos = function(tokens) {
68   var self = this;
69     return tokens[self.tkBeginPos + self.matchLength - 1].txtEndPos;
70 };
71
72 /**
73  * Sets the style class of the match segment.
74  * @function
75  * @param {(Number|String)} n - the style class to be applied
76  */
```

```
77 MatchSegment.prototype.setStyleClass = function(n) {
78   var self = this;
79   if (typeof n === 'number') {
80     self.styleClass = ['hl-', n % 10].join('');
81   }
82
83   if (typeof n === 'string') {
84     self.styleClass = n;
85   }
86 };
87
88 module.exports = MatchSegment;
```

Source file 18. src/js/simtexter/simtexter.js

```
1  /* jshint undef:true, unused:true, node:true, browser:true */
2  'use strict';
3
4  var $            = require('jQuery');
5  var XRegExp      = require('XRegExp');
6  var Match        = require('./match.js');
7  var MatchSegment = require('./matchSegment.js');
8  var Text         = require('./text.js');
9  var Token        = require('./token.js');
10
11 /**
12  * Creates an instance of {SimTexter}.
13  * @constructor
14  * @param {this}        SimTexter
15  * @param {Object}      storage   - the object that holds the app's settings
16  */
17 function SimTexter(storage) {
18   this.ignoreLetterCase  = storage.getItemValueByKey('ignoreLetterCase');
19   this.ignoreNumbers     = storage.getItemValueByKey('ignoreNumbers');
20   this.ignorePunctuation = storage.getItemValueByKey('ignorePunctuation');
21   this.replaceUmlaut     = storage.getItemValueByKey('replaceUmlaut');
22   this.minMatchLength    = storage.getItemValueByKey('minMatchLength');
23
24   this.texts             = [];
25   this.tokens            = [new Token()];
26   this.uniqueMatches     = 0;
27 }
28
29 /**
30  * Returns a promise that handles the comparison process.
31  * When resolved, an array of nodes is returned,
32  * which holds the text and the highlighted matches.
33  * @function
34  * @param {Array<InputText>} inputTexts - the array of {InputText} objects
35  *                                       which hold information about the user
36  *                                       input
37  */
38 SimTexter.prototype.compare = function(inputTexts) {
39   var self     = this,
40       deferred = $.Deferred(),
41       forwardReferences = [],
42       similarities = [];
43
44     // Read input (i.e. cleaning, tokenization)
45     self._readInput(inputTexts, forwardReferences);
46     // Get matches
47     similarities = self._getSimilarities(0, 1, forwardReferences);
48
```

```
49      if (similarities.length) {
50        // Return input string as HTML nodes
51        deferred.resolve(self._getNodes(inputTexts, similarities));
52      } else {
53        deferred.reject('No similarities found.');
54      }
55
56    return deferred.promise();
57  };
58
59  /**
60   * Applies a style class to each match segment
61   * and removes duplicates from the array of matches.
62   * Duplicates or overlapping segments can be traced,
63   * if one observes the target {MatchSegment} objects
64   * stored in the array matches.
65   * Sorting of matches by target {MatchSegment},
66   * with its tkBeginPos in ascending order
67   * and its matchLength in descending order,
68   * makes removal of duplicates easy to handle.
69   * The first {MatchSegment} with a given tkBeginPos
70   * has the longest length. All others with the same tkBeginPos
71   * have the same or a smaller length, and thus can be discarded.
72   * @function
73   * @private
74   * @param   {Array} matches - the array that holds the match segments,
75   *                            stored in pairs
76   * @returns {Array}         - the array of unique matches
77   */
78  SimTexter.prototype._applyStyles = function(matches) {
79    var self = this;
80
81    // Sort matches by target {MatchSegment},
82    // where tkBeginPos in ascending order and matchLength in descending order
83    var sortedMatches = self._sortSimilarities(matches, 1);
84    var sortedMatchesLength = sortedMatches.length;
85    var styleClassCnt = 1;
86
87    // Add first match in array of unique matches to have a starting point
88    var uniqueMatch = [sortedMatches[0][0], sortedMatches[0][1]];
89    uniqueMatch[0].setStyleClass(0);
90    uniqueMatch[1].setStyleClass(0);
91    var aUniqueMatches = [uniqueMatch];
92
93    // For each match in sortedMatches[]
94    for (var i = 1; i < sortedMatchesLength; i++) {
95      var lastUniqueMatch = aUniqueMatches[aUniqueMatches.length - 1][1];
96      var match = sortedMatches[i][1];
97
98      // If not duplicate
99      if (lastUniqueMatch.tkBeginPos != match.tkBeginPos) {
100       // if not overlapping
101       if (lastUniqueMatch.getTkEndPosition() - 1 < match.tkBeginPos) {
102         uniqueMatch = [sortedMatches[i][0], sortedMatches[i][1]];
103         uniqueMatch[0].setStyleClass(styleClassCnt);
104         uniqueMatch[1].setStyleClass(styleClassCnt);
105         aUniqueMatches.push(uniqueMatch);
106         styleClassCnt++;
107       } else {
108         // end-to-start overlapping
109         // end of lastUniqueMatch overlaps with start of match
110         if (lastUniqueMatch.getTkEndPosition() < match.getTkEndPosition()) {
111           var styleClass = ( /overlapping$/.test(lastUniqueMatch.styleClass) ) ?
                  lastUniqueMatch.styleClass : lastUniqueMatch.styleClass + '
```

```
                overlapping';
112             // Overwrite the style of the last unique match segment
113             // and change its length accordingly
114             aUniqueMatches[aUniqueMatches.length - 1][0].setStyleClass(styleClass)
                    ;
115             aUniqueMatches[aUniqueMatches.length - 1][1].setStyleClass(styleClass)
                    ;
116             aUniqueMatches[aUniqueMatches.length - 1][1].matchLength = match.
                    tkBeginPos - lastUniqueMatch.tkBeginPos;
117
118             // Add the new match segment
119             uniqueMatch = [sortedMatches[i][0], sortedMatches[i][1]];
120             uniqueMatch[0].setStyleClass(styleClass);
121             uniqueMatch[1].setStyleClass(styleClass);
122             aUniqueMatches.push(uniqueMatch);
123           }
124         }
125       }
126     }
127
128     self.uniqueMatches = aUniqueMatches.length;
129     return aUniqueMatches;
130 };
131
132 /**
133  * Returns a regular expression depending on the comparison options set.
134  * Uses the XRegExp category patterns.
135  * @function
136  * @private
137  * @returns {XRegExp} - the regular expression
138  */
139 SimTexter.prototype._buildRegex = function() {
140     var self = this,
141         // XRegExp patterns
142         NUMBERS     = '\\p{N}',
143         PUNCTUATION = '\\p{P}',
144         regex       = '';
145
146     if (self.ignoreNumbers) {
147       regex += NUMBERS;
148     }
149
150     if (self.ignorePunctuation) {
151       regex += PUNCTUATION;
152     }
153
154     return (regex.length > 0) ? XRegExp('[' + regex + ']', 'g') : undefined;
155 };
156
157 /**
158  * Cleans the input string according to the comparison options set.
159  * @function
160  * @private
161  * @param   {String} inputText - the input string
162  * @returns {String}           - the cleaned input string
163  */
164 SimTexter.prototype._cleanInputText = function(inputText) {
165     var self = this,
166         text = inputText;
167
168     var langRegex = self._buildRegex();
169
170     if (langRegex) {
171       text = inputText.replace(langRegex, ' ');
```

```
172     }
173
174     if (self.ignoreLetterCase) {
175       text = text.toLowerCase();
176     }
177
178     return text;
179   };
180
181   /**
182    * Returns a "cleaned" word, according to the comparison options set.
183    * @function
184    * @private
185    * @param   {String} word - a sequence of characters, separated by one
186    *                          or more white space characters (space, tab, newline)
187    * @returns {String}      - the cleaned word
188    */
189   SimTexter.prototype._cleanWord = function(word) {
190     var self = this,
191         umlautRules = {
192           'ä': 'ae',
193           'ö': 'oe',
194           'ü': 'ue',
195           'ß': 'ss',
196           'æ': 'ae',
197           'œ': 'oe',
198           'Ä': 'AE',
199           'Ö': 'OE',
200           'Ü': 'UE',
201           'Æ': 'AE',
202           'Œ': 'OE'
203         },
204         token = word;
205
206     if (self.replaceUmlaut) {
207       token = word.replace(/ä|ö|ü|ß|æœ||Ä|Ö|Ü|ÆŒ|/g, function(key){
208         return umlautRules[key];
209       });
210     }
211
212     return token;
213   };
214
215   /**
216    * Finds the longest common substring in the source and the target text
217    * and returns the best match.
218    * @function
219    * @private
220    * @param   {Number} srcTxtIdx     - the index of the source text in texts[]
221    *                                   to be compared
222    * @param   {Number} trgTxtIdx     - the index of the target text in texts[]
223    *                                   to be compared
224    * @param   {Number} srcTkBeginPos - the index of the token in tokens[]
225    *                                   at which the comparison should start
226    * @param   {Array}  frwReferences - the array of forward references
227    * @returns {Match}                - the best match
228    */
229   SimTexter.prototype._getBestMatch = function(srcTxtIdx, trgTxtIdx, srcTkBeginPos
       , frwReferences) {
230     var self = this,
231         bestMatch,
232         bestMatchTkPos,
233         bestMatchLength = 0,
234         srcTkPos = 0,
```

```
235       trgTkPos = 0;
236
237     for ( var tkPos = srcTkBeginPos;
238         (tkPos > 0) && (tkPos < self.tokens.length);
239         tkPos = frwReferences[tkPos]                    ) {
240
241       // If token not within the range of the target text
242       if (tkPos < self.texts[trgTxtIdx].tkBeginPos) {
243         continue;
244       }
245
246       var minMatchLength = (bestMatchLength > 0) ? bestMatchLength + 1 : self.
              minMatchLength;
247
248       srcTkPos = srcTkBeginPos + minMatchLength - 1;
249       trgTkPos = tkPos + minMatchLength - 1;
250
251       // Compare backwards
252       if ( srcTkPos < self.texts[srcTxtIdx].tkEndPos &&
253            trgTkPos < self.texts[trgTxtIdx].tkEndPos &&
254            (srcTkPos + minMatchLength) <= trgTkPos      ) { // check if they
                 overlap
255         var cnt = minMatchLength;
256
257         while (cnt > 0 && self.tokens[srcTkPos].text === self.tokens[trgTkPos].
              text) {
258           srcTkPos--;
259           trgTkPos--;
260           cnt--;
261         }
262
263         if (cnt > 0) {
264           continue;
265         }
266       } else {
267         continue;
268       }
269
270       // Compare forwards
271       var newMatchLength = minMatchLength;
272       srcTkPos = srcTkBeginPos + minMatchLength;
273       trgTkPos = tkPos + minMatchLength;
274
275       while ( srcTkPos < self.texts[srcTxtIdx].tkEndPos &&
276               trgTkPos < self.texts[trgTxtIdx].tkEndPos &&
277               (srcTkPos + newMatchLength) < trgTkPos    && // check if they
                    overlap
278               self.tokens[srcTkPos].text === self.tokens[trgTkPos].text ) {
279         srcTkPos++;
280         trgTkPos++;
281         newMatchLength++;
282       }
283
284       // Record match
285       if (newMatchLength >= self.minMatchLength && newMatchLength >
              bestMatchLength) {
286         bestMatchLength = newMatchLength;
287         bestMatchTkPos  = tkPos;
288         bestMatch = new Match(srcTxtIdx, srcTkBeginPos, trgTxtIdx, bestMatchTkPos,
              bestMatchLength);
289       }
290     }
291
292     return bestMatch;
```

```
293 };
294
295 /**
296  * Returns an array of HTML nodes, containing the whole text,
297  * together with the hightlighted matches.
298  * The text content of each node is retrieved by slicing the input text
299  * at the first (txtBeginPos) and the last (txtEndPos) character position
300  * of each match.
301  * @function
302  * @private
303  * @param   {Array} inputTexts - the array of {InputText} objects,
304  *                               which hold information about each user input
305  * @param   {Array} matches    - the array that holds the {MatchSegment} objects
     ,
306  *                               stored in pairs
307  * @returns {Array}            - the array of HTML nodes,
308  *                               which holds the text and the highlighted
       matches
309  */
310 SimTexter.prototype._getNodes = function(inputTexts, matches) {
311   var self = this,
312       iTextsLength = inputTexts.length,
313       nodes = [];
314
315   var styledMatches = self._applyStyles(matches);
316
317   // For each input text
318   for (var i = 0; i < iTextsLength; i++) {
319     var inputText = inputTexts[i].text,
320         chIdx = 0,
321         chIdxLast = chIdx,
322         chEndPos = inputText.length,
323         mIdx = 0,
324         trgIdxRef = (i == 0) ? (i + 1) : (i - 1);
325         nodes[i] = [];
326
327     // Sort array of similarities
328     var sortedMatches = self._sortSimilarities(styledMatches, i);
329
330     // For each character position in input text
331     while (chIdx <= chEndPos) {
332       if (sortedMatches.length && mIdx < sortedMatches.length) {
333         var match = sortedMatches[mIdx][i];
334         // Get start character position of match segment
335         var mTxtBeginPos = match.getTxtBeginPos(self.tokens);
336         // Get end character position of match segment
337         var mTxtEndPos = match.getTxtEndPos(self.tokens);
338
339         // Create text node
340         var textNodeStr = inputText.slice(chIdxLast, mTxtBeginPos);
341         var textNode = document.createTextNode(textNodeStr);
342         nodes[i].push(textNode);
343
344         // Create link node for match segment
345         var linkNodeStr = inputText.slice(mTxtBeginPos, mTxtEndPos);
346         var linkNode = match.createLinkNode(linkNodeStr, sortedMatches[mIdx][
              trgIdxRef]);
347         nodes[i].push(linkNode);
348
349         mIdx++;
350         chIdx = mTxtEndPos;
351         chIdxLast = chIdx;
352       } else {
353         var lastTextNodeStr = inputText.slice(chIdxLast, chEndPos);
```

```
354         var lastTextNode = document.createTextNode(lastTextNodeStr);
355         nodes[i].push(lastTextNode);
356         chIdx = chEndPos;
357         break;
358       }
359       chIdx++;
360     }
361   }
362
363   return nodes;
364 };
365
366 /**
367  * Returns an array of matches,
368  * where each match is an array of two {MatchSegment} objects, stored in pairs.
369  * At index 0, the source {MatchSegment} object is stored,
370  * and at index 1, the target {MatchSegment} object.
371  * @function
372  * @param   {Number} srcTxtIdx     - the index of the source {Text} object
373  *                                   in texts[] to be compared
374  * @param   {Number} trgTxtIdx     - the index of the target {Text} object
375  *                                   in texts[] to be compared
376  * @param   {Array}  frwReferences - the array of forward references
377  * @returns {Array}                - the array that holds the {MatchSegment}
378  *                                   objects, stored in pairs
379  */
380 SimTexter.prototype._getSimilarities = function(srcTxtIdx, trgTxtIdx,
        frwReferences) {
381   var self         = this,
382       similarities = [],
383       srcTkPos     = self.texts[srcTxtIdx].tkBeginPos,
384       srcTkEndPos  = self.texts[srcTxtIdx].tkEndPos;
385
386   while ((srcTkPos + self.minMatchLength) <= srcTkEndPos) {
387     var bestMatch = self._getBestMatch(srcTxtIdx, trgTxtIdx, srcTkPos,
          frwReferences);
388
389     if (bestMatch && bestMatch.matchLength > 0) {
390       similarities.push([
391           new MatchSegment(bestMatch.srcTxtIdx, bestMatch.srcTkBeginPos,
                bestMatch.matchLength),
392           new MatchSegment(bestMatch.trgTxtIdx, bestMatch.trgTkBeginPos,
                bestMatch.matchLength)
393         ]);
394       srcTkPos += bestMatch.matchLength;
395     } else {
396       srcTkPos++;
397     }
398   }
399
400   return similarities;
401 };
402
403 /**
404  * Creates the forward reference table.
405  * @function
406  * @private
407  * @param {Text}   text          - a {Text} object
408  * @param {Array}  frwReferences - the array of forward references
409  * @param {Object} mtsTags       - the hash table of minMatchLength
410  *                                 sequence of tokens (MTS)
411  */
412 SimTexter.prototype._makeForwardReferences = function(text, frwReferences,
        mtsTags) {
```

```
413   var self      = this,
414     txtBeginPos = text.tkBeginPos,
415     txtEndPos   = text.tkEndPos;
416
417   // For each token in tokens[]
418   for (var i = txtBeginPos; (i + self.minMatchLength - 1) < txtEndPos; i++) {
419     // Concatenate tokens of minimum match length
420     var tag = self.tokens.slice(i, i + self.minMatchLength).map(function(token)
          {
421       return token.text;
422     }).join('');
423
424     // If hash table contains tag
425     if (tag in mtsTags) {
426       // Store current token position at index mtsTags[tag]
427       frwReferences[mtsTags[tag]] = i;
428     }
429     // Add tag to hash table and assign current token position to it
430     mtsTags[tag] = i;
431   }
432 };
433
434 /**
435  * Reads the input string, and initializes texts[] and tokens[].
436  * Creates also the forward reference table.
437  * @function
438  * @private
439  * @param {Array} inputTexts    - the array of {InputText} objects
440  *                                that hold information on the user input
441  * @param {Array} frwReferences - the array of forward references
442  */
443 SimTexter.prototype._readInput = function(inputTexts, frwReferences) {
444   var self         = this,
445       mtsHashTable = {},
446       iLength      = inputTexts.length;
447
448   for (var i = 0; i < iLength; i++) {
449     var inputText = inputTexts[i];
450     // Compute text's words
451     var nrOfWords = inputText.text.match(/[^\s]+/g).length;
452     // Initialize texts[]
453     self.texts.push(new Text(inputText.mode, inputText.text.length, nrOfWords,
          inputText.fileName, self.tokens.length));
454     // Initialize tokens[]
455     self._tokenizeInput(inputText.text);
456     // Update text's last token position
457     self.texts[i].tkEndPos = self.tokens.length;
458     // Create array of forward references
459     self._makeForwardReferences(self.texts[i], frwReferences, mtsHashTable);
460   }
461 };
462
463 /**
464  * Sorts matches by source or target {MatchSegment},
465  * depending on the idx value.
466  * @function
467  * @private
468  * @param   {Array}  matches - the array of matches to be sorted
469  * @param   {Number} idx     - the index of the array of
470  *                             the {MatchSegment} objects
471  * @returns {Array}          - the sorted array of matches
472  */
473 SimTexter.prototype._sortSimilarities = function(matches, idx) {
474   var sortedSims = matches.slice(0);
```

```
475
476   sortedSims.sort(function(a, b) {
477     var pos = a[idx].tkBeginPos - b[idx].tkBeginPos;
478     if (pos) {
479       return pos;
480     }
481     return b[idx].matchLength - a[idx].matchLength;
482   });
483
484   return sortedSims;
485 };
486
487 /**
488  * Tokenizes the input string.
489  * @param {Object} inputText - the input string to be tokenized
490  */
491 SimTexter.prototype._tokenizeInput = function(inputText) {
492   var self        = this,
493       wordRegex = /[^\s]+/g,
494       match;
495
496   var cleanedText = self._cleanInputText(inputText);
497
498   while (match = wordRegex.exec(cleanedText)) {
499     var word = match[0];
500     var token = self._cleanWord(word);
501
502     if (token.length > 0) {
503       var txtBeginPos = match.index;
504       var txtEndPos   = match.index + word.length;
505       // Add token to tokens[]
506       self.tokens.push(new Token(token, txtBeginPos, txtEndPos));
507     }
508   }
509 };
510
511 module.exports = SimTexter;
```

Source file 19. src/js/simtexter/text.js

```
1  /* jshint undef:true, unused:true, node:true, browser:true */
2  'use strict';
3
4  /**
5   * Creates an instance of a {Text},
6   * which holds information on the input string.
7   * @constructor
8   * @this  {Text}
9   * @param {String} inputMode      - the mode of the input (i.e. 'File'
10  *                                  or 'Text')
11  * @param {Number} nrOfCharacters - the total number of characters
12  *                                  of the input string
13  * @param {Number} nrOfWords      - the total number of words
14  *                                  of the input string
15  * @param {String} fileName       - the name of the file
16  * @param {Number} tkBeginPos     - the index (inclusive) of the token
17  *                                  in {SimTexter.tokens[]}, at which
18  *                                  the input string starts
19  * @param {Number} tkEndPos       - the index (non-inclusive) of the token
20  *                                  in {SimTexter.tokens[]}, at which
21  *                                  the input string ends
22  */
23 function Text(inputMode, nrOfCharacters, nrOfWords, fileName, tkBeginPos,
```

```
    tkEndPos) {
  this.inputMode      = inputMode;
  this.fileName       = fileName;
  this.tkBeginPos     = tkBeginPos     || 0;
  this.tkEndPos       = tkEndPos       || 0;
  this.nrOfCharacters = nrOfCharacters || 0;
  this.nrOfWords      = nrOfWords      || 0;
}

module.exports = Text;
```

Source file 20. src/js/simtexter/token.js

```
/* jshint undef:true, unused:true, node:true, browser:true */
'use strict';

/**
 * Creates an instance of a {Token}.
 * A {Token} records the starting and ending character position
 * of a word in the input string, to facilitate reconstruction of the input
 * during output of the comparison results.
 * A word is a sequence of characters,
 * separated by one or more whitespaces or newlines.
 * The text of the {Token} corresponds to the "cleaned" version of a word.
 * All characters, as defined by the comparison options set by the user,
 * are removed/replaced from the token's text.
 * @constructor
 * @this  {Token}
 * @param {String} text        - the text of the word after being "cleaned"
 *                               according to the comparison options
 *                               set by the user
 * @param {Number} txtBeginPos - the index of the word's first character
 *                               (inclusive) in the input string
 * @param {Number} txtEndPos   - the index of the word's last character
 *                               (non-inclusive) in the input string
 */
function Token(text, txtBeginPos, txtEndPos) {
  this.text        = text        || '';
  this.txtBeginPos = txtBeginPos || 0;
  this.txtEndPos   = txtEndPos   || 0;
}

module.exports = Token;
```

Bibliography

Charras, Christian & Lecroq, Thierry (1997). *Handbook of Exact String-Matching Algorithms*. Retrieved 14 February 2016, from http://www-igm.univ-mlv.fr/~lecroq/string/string.pdf.

ECMA International (2012, December). *ECMA-376: Office Open XML File Formats - Fundamentals and Markup Language Reference Office Open XML* (4th ed.). Retrieved 16 February 2016, from http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-376,%20Fourth%20Edition,%20Part%201%20-%20Fundamentals%20And%20Markup%20Language%20Reference.zip.

Goodrich, Michael. T. & Tamassia, Roberto (2010). *Data Structures & Algorithms in Java* (5th ed.). New York: J. Wiley & Sons.

Grune, Dick (n.d.). *A concise report on the algorithms in Sim*. Retrieved 7 October 2015, from http://dickgrune.com/Programs/similarity_tester/TechnReport.

Grune, Dick & Huntjens, Martin (1989). *Detecting copied submissions in computer science workshops*. Amsterdam: Faculteit Wiskunde & Informatica, Vrije Universiteit. Retrieved 7 October 2015, from http://dickgrune.com/Programs/similarity_tester/Paper.ps.

Grune, Dick (2015). *Sim(1): Unix-style manual page*. Retrieved 7 October 2015, from http://dickgrune.com/Programs/similarity_tester/sim.pdf.

Jain, Sanket & Pandey, Manish (2012). Hash Table Based Word Searching Algorithm. In: *International Journal of Computer Science and Information Technologies*, Vol. 3 (3), pp. 4385-4388. Retrieved 19 October 2015, from http://www.ijcsit.com/docs/Volume3/vol3Issue3/ijcsit20120303116.pdf.

Lowdermilk, Travis (2013). *User-Centered Design*. Sebastopol: O'Reilly.

Mishr, Sattyam Kishor & Pande, Manish (2010). An Efficient Word Matching Algorithm For off Line Text. In: *Global Journal of Computer Science and Technology*, Vol. 10 (11), pp. 23-28. Retrieved 20 October 2015, from http://globaljournals.org/GJCST_Volume10/4-An-Efficient-Word-Matching-Algorithm-For-off-Line-Text.pdf.

OASIS Open (2011, September 29). *OASIS: Open Document Format for Office Applications (OpenDocument) Version 1.2 - Part 1: OpenDocument Schema*. Retrieved 16 February 2016, from http://docs.oasis-open.org/office/v1.2/os/OpenDocument-v1.2-os-part1.pdf.

Sommerville, Ian (2011). *Software engineering* (9th ed.). Boston: Pearson.

W3C (2015, April 21). *File API: W3C Working Draft*. Retrieved 3 February 2016, from https://www.w3.org/TR/FileAPI/.

Weber-Wulff, Debora (2014). *False Feathers: A Perspective on Academic Plagiarism*. Berlin: Springer Berlin.

Weber-Wulff, Debora (2014, September 2). Homebrew collusion detection. In: *Copy, Shake, and Paste* [Blog]. Retrieved 16 October 2015, from http://copy-shake-paste.blogspot.de/2014/09/homebrew-collusion-detection.html.

Affidavit

I hereby confirm that I have written the present thesis independently and have used no aids other than those stipulated. All direct or indirect quotations taken from external sources are marked as such.

The present thesis has neither been published nor submitted to another examination authority in the same or similar form.

Eidesstattliche Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe. Alle aus fremden Quellen im Wortlaut oder dem Sinn nach entnommenen Aussagen sind durch Angaben der Herkunft kenntlich gemacht.

Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungskommission vorgelegt und auch nicht veröffentlicht.

Berlin, March 7, 2016	Sofia Kalaidopoulou
DATE | DATUM	SIGNATURE | UNTERSCHRIFT