Seshat – A sync system for Audiobooks and eBooks

Adnan Dervisevic      Tobias Oskarsson
Seshat – A sync system for Audiobooks and eBooks

Summary

In this degree project we present a way to construct a synchronization system that is able to create a timings file, which is the file the system uses to know how to sync the eBook and audiobook, using speech recognition and estimation algorithms. This file is then used by the system to let the user select a sentence and have the audiobook start reading from that sentence, or vice versa. This system can create these files with a mean offset from a manually timed file which is within our expectations for the system.

We use estimation algorithms to fill in the blanks where the speech recognition falls short. Speech recognition accuracy is typically between 40-60%, sometimes dipping lower, so there are blanks to fill in. Using basic algebraic principles of calculating velocity we can extrapolate the speed of a reader, using the duration of the audiobook as the time and the amount of characters written as the distance. For increased accuracy we derive this value on a per-chapter basis.

Using this method we are able to create accurate files, which the user can use to freely sync any location in the book. Our system is designed to work for any book in the world that does not have an audiobook which cuts off between sentences in the audio files.

We manually create timings files for four different books with widely varying publishing dates, author styles, reader style and gender to create as wide and representative a testing pool as possible for the project.

Date: 16th of June, 2014
Author: Adnan Dervisevic, Tobias Oskarsson
Examiner: Hanna Aknouche-Martinsson
Advisor: Dena Hussain
Programme: Computer Engineering and System Development, 180 HE credits
Main field of study: Computer Engineering
Education level: First cycle
Course code: EDT500, 15 HE credits
Keywords: Speech recognition, Audiobooks, eBooks, EPUB, MP3, XML, Synchronization, Sync
Publisher: University West, Department of Engineering Science
SE-461 86 Trollhättan, Sweden
Phone: + 46 520 22 30 00, Fax: + 46 520 22 32 99, www.hv.se
Preface

Thanks to our parents for supporting us and Denis Dervisevic for his advice and support. We would also like to thank our supervisor Dena Hussain for her advice throughout the process.

We used an agile Extreme Programming approach to creating the system, using Pair Programming. Adnan Dervisevic focused on the user interface, estimation algorithms and the book structure. Tobias was more inclined towards the speech recognition, audio implementation, the comparison engine and manual timing. We spread out the reading for the manual timings files evenly between us.
# Table of contents

1. Introduction .......................................................................................................................... 1

2. Background/Theory .............................................................................................................. 1
   2.1 Speech Recognition ........................................................................................................... 1
   2.2 The Timings File ............................................................................................................... 2
   2.3 EPUB ................................................................................................................................. 3

3. Methodology .......................................................................................................................... 3
   3.1 Methods Compared ......................................................................................................... 3
      3.1.1 Speech recognition & Mathematical Approximation ................................................ 4
      3.1.2 Manual ...................................................................................................................... 4
   3.2 The Books ......................................................................................................................... 4

4. Implementation ...................................................................................................................... 6
   4.1 Estimation Algorithm ....................................................................................................... 6
   4.2 How the timings files are created using the speech recognition engine ......................... 6
   4.3 Framework ....................................................................................................................... 7
   4.4 File management ............................................................................................................. 8
   4.5 EPUB Reader .................................................................................................................. 9
   4.6 Audio ............................................................................................................................... 9
   4.7 Synchronization ............................................................................................................. 10
   4.8 Timings file ..................................................................................................................... 11
   4.9 Comparison ..................................................................................................................... 11

5. Results .................................................................................................................................. 13
   5.1 Timing Accuracy ............................................................................................................. 13
      5.1.1 Speech Recognition .................................................................................................. 13
      5.1.2 The Hunger Games ................................................................................................ 15
      5.1.3 Casino Royale .......................................................................................................... 17
      5.1.4 Five Go Off in a Caravan ........................................................................................ 19
      5.1.5 Eric .......................................................................................................................... 21
   5.2 Performance ..................................................................................................................... 22
      5.2.1 Time to create a timings file ................................................................................... 22
Appendices

A. Removal of inaccurate timings
B. Filling missing values
C. Compare two books and calculates the differences
D. System Normal Use
E. Manual Timing
F. Comparison window
G. Complete flowchart for opening a book
H. Complete source code

Glossary

<table>
<thead>
<tr>
<th>EPUB</th>
<th>Electronic Publication</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chapter structure</td>
<td>A book with audio files for every single chapter</td>
</tr>
<tr>
<td>Non-chapter structure</td>
<td>A book which does not have audio files for every chapter</td>
</tr>
<tr>
<td>Timings File</td>
<td>A file containing timing for every sentence in the book for syncing.</td>
</tr>
</tbody>
</table>
1 Introduction

Seshat, named after the Ancient Egyptian goddess of wisdom, knowledge and libraries, is a sync system created to synchronize eBooks and Audiobooks. The idea is conceptually similar to Amazon’s Whispersync [1], but aims to make an open version that can handle more files. Whispersync is limited to a catalogue that Amazon has chosen, 15,000 books, while our system can handle most books. It allows you to highlight any sentence in a book and sync the audio to the position you are reading from. It also handles the reverse, allowing the book to catch up to where the audio file is.

There are many ways to approach a solution to the problem, but for the project we ended up using speech recognition, aided by mathematical approximation, and a manual system. This method allows us to get measurable data, as we create a base-line file for comparison, and then we are able to compare our other implementations against it.

One of the main objectives of this project was to obtain measurable data, which could be used for comparisons after the implementation. We chose to create manual timings files, which are similar to movie subtitles timings files, to be able to get these accuracy measurements, as we can compare the automatically created files against them and see how far off they are.

For the purposes of the project, we are able to use most audio files. Any structure of files is acceptable for our system, except where a sentence cuts off at the end of a file. We’ve limited ourselves to English books.

For the books that we match to the audio, the EPUB (Electronic Publication) format was chosen. It’s one of the most prolific formats for eBooks [2], and can be found in libraries as well [9]. It’s the standard recommended by the International Digital Publishing Forum as well, which ensures a measure of stability in the structure of the books [2, 8].

2 Background/Theory

2.1 Speech Recognition

The primary objective of a speech recognition engine is to listen for predefined commands from an input but it can also dictate what it hears, i.e. try to find out what the user says. The purpose of a speech recognition engine can be different depending on the application it is used in, but for this project we narrowed the scope to simply listening to sentences from the book that we create as objects for the system to listen for as we load the book in. As books only has one person reading, it is possible that accuracy could be improved by introducing self-learning elements to the system.

Something that is also crucial for our application with our chosen speech recognition engine is that it does not only return a timestamp, where the match was made, but also a duration for how long the sentence is. As seen in Figure 1, these values can then be used together to fill in a blank with complete accuracy, given that it is an accurate timestamp/duration. A duration is, however, not possible
to ascertain at this time and has to be estimated by the estimation algorithm, which is described in detail in the implementation section.

Figure 1. How a previous position creates an accurate next position, without a duration.

The chosen speech recognition engine also returns a statistical confidence value, which we found was not suitable for our applications and instead we use weighted values based on their statistical likelihood in accordance with their audio position. Speech recognition is currently still at a relatively low rate of accuracy, at least for use in areas such as this system, where a 100% accuracy is needed to use it without any additional algorithms to correct any errors or gaps.

The system provides confidence scores of the recognized sentences, but we have chosen to eschew those in favour of using our own algorithms to choose a speech hit closest to an estimated window. These algorithms can be seen in Appendix A, B and C.

The C# programming language, introduced by Microsoft in 2000, was the language chosen for the project. It is ideal for the purposes of this project as it comes with a highly built-in speech recognition engine. It is a speech engine which does not require an acoustics model & custom dictionary to work, and which can accept a stream as input instead of relying solely on the microphone input.

The speech recognition engine has a maximum limit of 1024 sentences and therefore the system must compensate for this by splitting bigger chapters in subchapters.

CMU Sphinx, which is popular speech recognition engine for research [11], was also considered for the project but it does not have all the features and ease of use the project needs.

2.2 The Timings File

A timings file is a file containing information on each sentence in the entire book, which allows us to sync the eBook and Audiobook. It acts as a reference, and contains a position for the audio for each sentence, how long it takes to read the sentence and which audio file the sentence is on. A sentence in Seshat always ends with a dot and if the last sentence in a chapter does not have a dot the sentence will end with the chapter. The reason behind this was to get as high success rate as possible with the speech recognition engine. For example if a sentence would have ended with either a dot or an exclamation mark, a sentence like ‘Hurray!, Hurray!, Hurray! Cheered Anne.’ would be divided into four different sentences and the speech recognition engine would have trouble finding them inside the audio because of the length of the sentences.

A Timings file also contains information that is used for the comparisons and displaying data that is of great interest to us in the timings file. We are able to parse what the success rate was of the speech recognition and the reading speed, as well as of course using the sentences to check differences in audio-position and length. We save this information not only on a book level, but also a
chapter level, to get an increased granularity of our comparisons. To make it possible to have multiple versions of the same book the timings file uses the MD5 checksum of the eBook file as the name of the timings file.

### 2.3 EPUB

EPUB is an open format for eBooks which is maintained by the International Digital Publishing Forum (IPDF), which aims to facilitate reflowable digital publications [2]. It utilizes many standards that are widely used and well-documented, and it maintains freely available specifications.

An EPUB eBook is at its core a compressed file, a zip file. Inside this file, with the file extension .epub, there’s a MIME (Multipurpose Internet Mail Extensions) type file, a directory with the title META-INF and another directory OEBPS (Open eBooks Publication Structure). The MIME file is what defines the EPUB for what it is. The META-INF directory is a requirement of the format, and is what holds the container.xml file. This is a crucial file, as it describes where we can find the root file for all the content. In the same folder as the root file, we can find the content that we are looking for. There is also a navMap (navigation map) in the EPUB book [6], which we use to navigate the chapters. The content is typically stored in the HTML/XHTML format and is what we use to display the book to the reader. It is also what we parse as an input for our speech recognition engine.

EPUB files are standardized by the IPDF organization [8] which ensures that we can expect a degree of consistency. There is, for example, always a toc nav (Table of Contents) element, which we are able to parse. Books, however, are not standardized. This presents a challenge, as many authors like to stylize their books via use of chapters and parts.

When parsing a book there are certain assumptions one can make based on the initial information one receives. If the book has a flat structure, one or more chapters containing sentences, then we can determine that a book is of a normal chapter structure. Books also come in different arrangements, such as a structure of several parts or multiple books containing chapters or parts.

### 3 Methodology

For the project C# was chosen as the programming language, for its built-in speech recognition engine. The C# speech recognition engine is not as highly configurable as some custom speech recognition engines are, but its ease of use and ability to use a stream as input is unmatched and a perfect fit for the project.

#### 3.1 Methods Compared

For the purposes of the project two ways of creating timings files were devised. The primary method we created for making timings files is a mixture of speech recognition and mathematical algorithms. It enables us to automatically get a complete timings file which is able to achieve a success-rate that we deem acceptable, as demonstrated in later chapters. To measure the accuracy of the
aforementioned method we have furthermore manually created timings files by listening and reading books at the same time, and timing them out. We are then able to compare them and see how large the difference is sentence by sentence.

3.1.1 Speech recognition & Mathematical Approximation
This method uses a proprietary mix of speech recognition and algebraic approximations, to find sentences and fill in the gaps where the speech recognition is unable to, to automatically create a timings file that is as close to perfect as we can get it. It makes a first pass through the file using only the speech recognition engine. The accuracy is not at levels where we can use purely speech recognition, so we have many gaps in our timings file. While we have tried to minimize this from the get-go, there is also a presence of false positives at this stage.

The time it takes to construct a timings file using speech recognition is heavily influenced by the audio structure. With a pure character structure every chapter just need to check against its own audio file while a non-chapter structure needs to check against all the audio files. The system reduces the time it takes for non-chapter structure by skipping to position of the last sentence in the previous chapter if possible.

3.1.2 Manual
Finally we manually timed entire books. This is a very time intensive process as most books are several hours in length, and would be the preferred method if it was imposed as an audiobook creation standard. For our purposes, however, it serves as a baseline, which we make comparisons to. It lets us get metrics such as accuracy percentage, mean time difference and more, which form the basis for demonstrating the accuracy of the other systems. As a basis for comparison it is not perfect, because a human does not wait the same amount of time every time after a sentence has ended to mark the end of a sentence. It is more inexact, and there will always be slight discrepancies.

This method has many similarities to how traditional subtitles for video media are created [10]. There are minor differences between the way subtitles are created in video media and our system, to appropriate it to be more fitting for books which do not have long periods of silence. In traditional subtitles you press “lead in” for subtitles to start appearing, and then “lead off” to make them stop. As there are many silences in movies, this system works fantastically, but for our system we chose to have only one button. We instead choose to interpret it in our system that when one sentence ends, the next starts. As the pace is very rapid, it would be very difficult for a user to use a lead in/lead out system for manual book timing.

The requirements for manual timing are very low, as it simply changes the interface and records key presses, which are then saved to a timings file.

3.2 The Books
During the early development, a few books were chosen for the project with unique qualities that introduced varying challenges to the system, with either unique structure or language. The primary

Tolkien’s seminal work, *The Fellowship of the Ring*, was chosen for having an audiobook with fairly low quality and somewhat archaic and difficult language. It also used a multi-book structure, which was used to build our parser. Martin’s *A Dance with Dragons* features a very modern recording but most interestingly has a chapter structure which only uses names from the characters as the title/enumerator. It presented a challenge for the chapter navigation, and helped us further develop it. *The Hunger Games*, lastly, had a different file structure from the others while being a modern book of moderate length, substantially shorter than the two other works mentioned.

For developing the detailed comparisons, however, manual timings files needed to be created. Both Tolkien and Martin’s works are too vast for the project, so *The Hunger Games* was chosen as the first book for manual timing, representing books over 10 hours. To create a wider and larger pool of manually timed books for comparison, the focus was then shifted to shorter books.

For the project two books were also chosen with a length spanning between four and five hours. Blyton, Enid, *Five Go Off in a Caravan* (1946) and Fleming, Ian, *Casino Royale* (1953). The shortest book chosen has a play time of 3½ hours and is Pratchett, Terry’s *Eric* (1990), unique among the books tested in the fact that it has no chapters.
4 Implementation

4.1 Estimation Algorithm

In addition to the speech recognition engine the system uses estimation or calculation to fill in the position and duration for sentences that are missing.

Our estimation algorithm uses a value we call CPS, characters per second, and multiplies it with the amount of characters in the sentence, this way we estimates a duration for a sentence.

The CPS value is calculated depending on the structure of the book. If the book is using a chapter structure the system calculates the CPS by taking the amount of characters in the current chapter and divides it with the length of the audio file for the chapter, if the book uses a non-chapter structure the system will use all the characters in the entire book and divide it with the complete length of the audiobook instead of only the current chapter.

4.2 How the timings files are created using the speech recognition engine

First the system loads in the first chapter and runs the speech recognizer with the given audio file. For every found sentence the system stores the timestamp and duration. Once we have a timings file full of the speech recognition timestamps, our algorithms are applied to sort out the errors. Using basic algebraic foundations, we can attempt to fill in blanks using the partially existing data and extrapolating from a few known facts. The estimation methods are, however, not only based on mathematics. They also use a system of ascertaining whether or not a sentence is a false positive.

To sort out the errors after the speech recognition, the system loops through all the sentences in the current chapter. For each loop the system does four things.

1. Locate a previous position that is valid, this is done by looping through all the previous sentences backwards and return the first position that is not equal to zero. If no position can be found, then the system automatically returns position zero.

2. Find the closest position to the previous in the list of possible positions for the current sentence.

3. Find the next valid position after the current sentence, this is done by looping through the sentences after the current one until it finds a position that is not equal to zero. When a position has been found the system estimates a three minutes interval where the next position should be in, now by checking if the next position is within this range the system knows if it is a valid position.

4. Check if the position found in step two is in between the previous and the next position, if it is then we know that the position is reasonable; otherwise the position is deemed incorrect and we set it to zero.

A code example of these four steps can be found in Appendix A.
Table 1 represents a fraction of a timings file after the speech recognition and to find out if the third sentence is a valid position or not the four steps above are used. During the third loop the system will return the position of first sentence as the previous position and fifth sentence as the next position. During the second step the system will select the 01:24:15 position instead because it is closest to the previous position. Because the selected position is in between the previous and next position the system knows that it is a valid position and therefore does not set it to zero.

<table>
<thead>
<tr>
<th>Sentences</th>
<th>Audio-positions</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sentence 1</td>
<td>01:24:05</td>
<td>True</td>
</tr>
<tr>
<td>Sentence 2</td>
<td>00:00:00</td>
<td>Not Found</td>
</tr>
<tr>
<td>Sentence 3</td>
<td>03:25:55 – 01:24:15</td>
<td>False – True</td>
</tr>
<tr>
<td>Sentence 4</td>
<td>00:00:00</td>
<td>Not Found</td>
</tr>
<tr>
<td>Sentence 5</td>
<td>01:24:35</td>
<td>True</td>
</tr>
</tbody>
</table>

Table 1. Part of a timings file after the speech recognition.

For the next part of the process, a variation of the traditional formula to calculate velocity is applied.

\[
velocity = \frac{distance}{time}
\]

The velocity in this case is the reader’s speed, on an individual character in a word basis, where velocity is expressed in characters per second. The distance is the amount of characters in the book, and the time is the duration of the audio file. The duration estimation is simply a case of counting the amount of characters in a given sentence, and then multiplying it by the time it takes to read a character.

When there is a sentence that is already known to be reasonable, we are able to use the duration and previous position to estimate the next position as seen previously in Figure 1. The position is thus assumed to be accurate, but there is no duration set using this method, where the previously discussed estimation algorithms become relevant. However, the longer one goes without a re-calibration point, i.e. a correct sentence, the lower the accuracy becomes. This is unavoidable, however, and is a problem with the method.

### 4.3 Framework

We created a custom eBook reader and an audio player for the sake of the project. We implemented it using ePubReader [3] and NAudio [4] for the reader and audio respectively. The system store the converted file in the memory for further use, to get around having to create temporary files.

We also created a program to measure accuracy etc. how close a speech estimated timings file are to the manually timed timings file (Appendix F). We also use Visual Studio’s built-in performance benchmark tools to highlight performance differences for the CPU usage.
4.4 File management

Figure 3 describes what happens when the user opens a new book in the system. The file open icon is among the first thing the user sees, and it remembers the last location a user opened a file from, for ease of use. After a book has been chosen, the system checks if there is an associated timings file.

If there is already a timings file present for the chosen book, it will attempt to automatically load in all the associated audio files and the last known location in the book. If the system is unable to find them, it will prompt the user to locate the audio files. It will then use an MD5 checksum to make sure it has all the correct files, and will not let you link incorrect files to the timings file, unless you choose to remove the file. If a timings file does not exist the user will be prompted to add audio files that go along with the chosen book. The user can choose any number of audio files, and they are automatically handled by the system after that.

To properly identify a chapter structure, the audio files must in some way match the chapters in the system. This is done by having them use the same file name as the chapter names. If the user chooses to not immediately add the audio files to the book, the user can add them later and simply use the system as a book reader.

When a new book has been open, the system will create a new tab for the chosen book. This tab stores everything about the book and all the settings, for example volume and current position. Every tab has its own thread that it uses to create the timings file and multithreading is used here to make it possible for the user to read another book while creating a timings file.

See Appendix H for a complete diagram for opening a book.

Figure 2. Opening a book and either loads the timings file and the associated audio files or prompts the user to select audio files.
4.5 EPUB Reader

After the user has chosen his book, we use the ePubReader [3] library to parse the contents of the EPUB file. As seen in Figure 2, we strip any custom CSS that is in the file, because in many cases it’s designed for smaller devices such as Kindles, and is not suitable for screen reading. A new tab is created in the program, which also links itself to the audio.

![Diagram of EPUB Reader process]

Figure 3. Opening a book and going through the structure of books, parts and chapters.

Most of the program is devoted to the book, to allow the user to easily read, maintaining only enough space for the audio controls, shown in Appendix D. Each book has its own tab, which maintains the reading positions for them. So there is no concern of losing the users position in the book when switching to peruse out the next one. It also maintains position over sessions, so when the user closes the software and opens it back up, they’ll find themselves where they left off.

If an EPUB book contains a nested structure, as in parts that contain chapters that contain sentences, we start using our algorithms for processing any amount of depth in a book structure. We are able at that point to determine that a book contains parts or sub-books that have a flat chapter structure and go through them accordingly. The last of the standard variation is a book containing several books that may also contain parts. In this particular arrangement you may have to parse three or more layers before you reach chapters containing sentences.

4.6 Audio

The audio is processed and loaded into memory using the NAudio [4] .NET library. If there is a timings file present, the user can use the player controls to skip backwards and forwards between chap-
ters. All the expected audio controls are also present, play, pause, stop, volume and a bar for changing the audio-position. For more precision in very long files we have also included an alternative way of skipping position, in which the user clicks the timer that shows current audio progress to choose an exact location.

For ease of use and the speech recognition engine, the system loads in all the previously designated audio files into the memory of the user’s computer. This means that however disjointed the original audio files are, the user sees a coherent full audio book in his progress bar. This also means it’s much easier to see how far along the reader is in the book they are currently reading, and how long it is.

### 4.7 Synchronization

The system can handle two types of synchronization, audio sync and book sync.

The audio sync works by searching through all sentence objects in the book until it finds a match. If a match cannot be made or multiple matches are made, it responds with an error message, and prompts the user to try again. When a sentence object is matched, it’s used as a reference in the timings file, where an audio start position is found. The audio file then automatically starts playing from that location, giving the user a seamless experience.

The book sync synchronization uses the currently playing or paused audio’s position to find a match in the timings file. The system sends this timestamp back to the code, which then checks through the entire timings file, finding which sentence has an audio position which the current position exceeds. For example, if a file is currently at 02:34:05, it will cycle through every sentence until it finds one which has an audio position greater than 02:34:05. When the system has found the exceeding position, it will save the text from the sentence and then use it to search for the selected sentence in the book to find the appropriate location for the user.

The user has two buttons used for synchronization when there is a timings file present. One of them makes the user able to skip forward/backwards to an audio position based on the current selection. There are tooltips explaining this in the software. Conversely there is another that does the opposite, and syncs the book to the audio location, whether that is behind or ahead of the current selection. The iconography represents this through having the sync button that uses the audio file progress as a basis by a musical note, and the other, which uses a sentence you selected, with letters. The icons can be seen in Figure 4.

Figure 4. Left is to sync the book from an audio position, the right is to sync the audio from a chosen sentence.
4.8 Timings file

Choosing to create timings files is done through menus selections. The speech recognition estimated timings process changes the UI to include a progress bar near the top of the program. It does not affect the user's functionality in any way, as it is still possible to read the book and listen to the file. Each timing runs on its own thread, so it’s possible switch tabs to read or listen to other files while it is timing.

If the user selects manual timing the interface changes, seen in Appendix E, to an interface that is mainly driven by pre-defined hotkeys. It is not possible to create new timings if a file already exists. To make it easier for the person creating the manual timings file, we highlight what counts as a sentence in our system. It is possible to save a progress file when creating a manual timing's file to prevent the user from having to finish the timings file in one session.

It was decided that the best choice for the timings file would be the XML (Extensible Markup Language) format. XML is a very flexible language [10], which is a great fit for our system with its uncertainties about structure et cetera. It allows us to pack in a lot of information in a small package.

In Figure 5 we can see a shortened excerpt of the timings file for the Casino Royale book.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Books>
  <Book
    CompletedTime="00:39:16.6755285"
    Successrate="65"
    CharactersPerSecond="12.8123463009572"
    Volume="0.5"
    CurrentAudioFileIndex="0"
    CurrentAudioPosition="00:00:00.0000000"
    ScrollValue="0">
    <AudioFiles>
      <AudioFile
        Path="C:\AudioBooks\Casino Royale.mp3"
        Checksum="4E-8D-5D-C9-6A-03-81-C1-CE-C3-F5-3D-D2-83-77-DD" />
    </AudioFiles>
    <Chapters>
      <Chapter
        Successrate="70"
        AudioFileIndex="0"
        AudioPosition="00:01:18.2100000"
        Duration="00:00:04.5000000" />
    </Chapters>
  </Book>
</Books>
```

Figure 5. A shortened excerpt of the Casino Royale timings file.

4.9 Comparison

The system also contains an option for comparing timings files. It has a simple interface in which the user selects two timings files and it checks them for differences. It has a baseline file, File A, which File B is checked against (Appendix C). If File B has a timing that is exactly one second earlier than File A's timing on the same line in the XML document, it is denoted as a -1 in the .difference file that is output by the system. Because of the inherent incompatibility for exact timing between a
computer and a human, we’ve chosen to have the differences highlighted in 500ms gaps. It is detailed to a point where you can easily see the differences visually, but without having an overwhelming amount of data ranges.

The difference file splits up the sentence comparisons on a per-chapter basis, and features means and median values as well. As well as these means for the chapters, it also has a mean/median for the whole book, which allows us to compare the book’s overall success rate to other books.
5 Results

5.1 Timing Accuracy

5.1.1 Speech Recognition

Typically a mean accuracy of English speech recognition can be found to be around 60% [7]. This accuracy is usable for many applications of speech recognition, but not for our system, which needs to have reasonable values on each sentence. It is still, however, an intrinsically valuable part of the system.

The way the system measures success rate is that after the system has removed all the inaccurate values, explained in 4.1, it will loop through all the sentences in the entire book and check if it has a value. This will then give a success rate for the speech recognition and not the entire book.

As shown in Figure 7, speech accuracy in our system varies, but is typically 40-60%.

![Speech Recognition Success Rate](image)

Figure 6. Speech accuracy for an entire book.

There is also variance on a per-chapter basis for the success rate of the speech recognition engine, demonstrated in Figure 8a. *A Dance with Dragons* represented on its own in 8b for its vast length.

a)
We compared the manual and speech recognized timings files to get offset values, these can be seen in Table 2. It compares each sentence of the speech recognized file to the manually timed files, and adds them all up to give us mean and median offsets to the manual timings files. The closer to zero, the better.

<table>
<thead>
<tr>
<th>Manual &amp; Speech</th>
<th>Casino Royale</th>
<th>Five Go Off in a Caravan</th>
<th>The Hunger Games</th>
<th>Eric</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean offset:</td>
<td>-0.337 s</td>
<td>0.622 s</td>
<td>0.132 s</td>
<td>0.851 s</td>
</tr>
<tr>
<td>Median offset:</td>
<td>0.142 s</td>
<td>0.231 s</td>
<td>-0.208 s</td>
<td>0.177 s</td>
</tr>
</tbody>
</table>

Table 2. Measured offsets to the manual timings files using speech recognition. Time in seconds.
5.1.2 The Hunger Games

5.1.2.1 Manual and Speech Recognition

*The Hunger Games* is the longest book of the ones we made manual timings files for, with a playtime of 11:13:06.

In Figure 9 we demonstrate the accuracy of the whole book, when comparing the manual timings with the speech recognition. An error is defined as being the absolute value of an offset from a manual timings file.

![Errors for the whole book with speech recognition](image)

Figure 8. Sentences error from 0 in ranges of 0.5 seconds.
In Table 3 we demonstrate the mean and median of each chapter in the book when comparing the manual timings with the speech recognition.

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Mean</th>
<th>Median</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chapter 1</td>
<td>1.232</td>
<td>0.473</td>
</tr>
<tr>
<td>Chapter 2</td>
<td>0.239</td>
<td>0.727</td>
</tr>
<tr>
<td>Chapter 3</td>
<td>4.927</td>
<td>0.08</td>
</tr>
<tr>
<td>Chapter 4</td>
<td>-0.213</td>
<td>0.028</td>
</tr>
<tr>
<td>Chapter 5</td>
<td>-0.539</td>
<td>0.054</td>
</tr>
<tr>
<td>Chapter 6</td>
<td>-0.675</td>
<td>-0.332</td>
</tr>
<tr>
<td>Chapter 7</td>
<td>2.893</td>
<td>2.64</td>
</tr>
<tr>
<td>Chapter 8</td>
<td>-0.554</td>
<td>-0.248</td>
</tr>
<tr>
<td>Chapter 9</td>
<td>-0.846</td>
<td>-0.298</td>
</tr>
<tr>
<td>Chapter 10</td>
<td>-0.656</td>
<td>-0.363</td>
</tr>
<tr>
<td>Chapter 11</td>
<td>-0.504</td>
<td>-0.416</td>
</tr>
<tr>
<td>Chapter 12</td>
<td>-0.485</td>
<td>-0.642</td>
</tr>
<tr>
<td>Chapter 13</td>
<td>-0.667</td>
<td>-0.729</td>
</tr>
<tr>
<td>Chapter 14</td>
<td>-0.271</td>
<td>-0.734</td>
</tr>
<tr>
<td>Chapter 15</td>
<td>-0.852</td>
<td>-0.755</td>
</tr>
<tr>
<td>Chapter 16</td>
<td>0.625</td>
<td>0.478</td>
</tr>
<tr>
<td>Chapter 17</td>
<td>-0.364</td>
<td>-0.45</td>
</tr>
<tr>
<td>Chapter 18</td>
<td>-0.311</td>
<td>-0.262</td>
</tr>
<tr>
<td>Chapter 19</td>
<td>-0.74</td>
<td>-0.258</td>
</tr>
<tr>
<td>Chapter 20</td>
<td>-0.653</td>
<td>-0.267</td>
</tr>
<tr>
<td>Chapter 21</td>
<td>-1.262</td>
<td>-0.257</td>
</tr>
<tr>
<td>Chapter 22</td>
<td>-1.069</td>
<td>-0.337</td>
</tr>
<tr>
<td>Chapter 23</td>
<td>-0.827</td>
<td>-0.347</td>
</tr>
<tr>
<td>Chapter 24</td>
<td>-0.792</td>
<td>-0.613</td>
</tr>
<tr>
<td>Chapter 25</td>
<td>-0.81</td>
<td>0.083</td>
</tr>
<tr>
<td>Chapter 26</td>
<td>-0.131</td>
<td>0.059</td>
</tr>
<tr>
<td>Chapter 27</td>
<td>6.054</td>
<td>4.96</td>
</tr>
</tbody>
</table>

Table 3. Time offsets from manual timing in seconds on a per-chapter basis.
5.1.3 Casino Royale

5.1.3.1 Manual and Speech Recognition

_Casino Royale_ is of medium length, with a playtime of 5 hours.

In Figure 10 we demonstrate the accuracy of the whole book, when comparing the manual timings with the speech recognition.

Figure 9. Sentences error from 0 in ranges of 0.5 seconds.
In Table 4 we demonstrate the mean and median of each chapter in the book when comparing the manual timings with the speech recognition.

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Mean</th>
<th>Median</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chapter 1</td>
<td>-1.89</td>
<td>0.275</td>
</tr>
<tr>
<td>Chapter 2</td>
<td>-0.882</td>
<td>-0.006</td>
</tr>
<tr>
<td>Chapter 3</td>
<td>-0.387</td>
<td>0.049</td>
</tr>
<tr>
<td>Chapter 4</td>
<td>-0.226</td>
<td>0.009</td>
</tr>
<tr>
<td>Chapter 5</td>
<td>0.064</td>
<td>0.199</td>
</tr>
<tr>
<td>Chapter 6</td>
<td>-0.092</td>
<td>0.149</td>
</tr>
<tr>
<td>Chapter 7</td>
<td>-0.441</td>
<td>0.089</td>
</tr>
<tr>
<td>Chapter 8</td>
<td>-0.388</td>
<td>0.052</td>
</tr>
<tr>
<td>Chapter 9</td>
<td>0.155</td>
<td>0.162</td>
</tr>
<tr>
<td>Chapter 10</td>
<td>-0.016</td>
<td>0.237</td>
</tr>
<tr>
<td>Chapter 11</td>
<td>-0.069</td>
<td>0.184</td>
</tr>
<tr>
<td>Chapter 12</td>
<td>-0.389</td>
<td>-0.066</td>
</tr>
<tr>
<td>Chapter 13</td>
<td>0.203</td>
<td>0.114</td>
</tr>
<tr>
<td>Chapter 14</td>
<td>-0.216</td>
<td>0.074</td>
</tr>
<tr>
<td>Chapter 15</td>
<td>0.085</td>
<td>0.129</td>
</tr>
<tr>
<td>Chapter 16</td>
<td>0.076</td>
<td>0.164</td>
</tr>
<tr>
<td>Chapter 17</td>
<td>-0.141</td>
<td>0.149</td>
</tr>
<tr>
<td>Chapter 18</td>
<td>-0.103</td>
<td>0.194</td>
</tr>
<tr>
<td>Chapter 19</td>
<td>-0.297</td>
<td>0.154</td>
</tr>
<tr>
<td>Chapter 20</td>
<td>-2.696</td>
<td>0.114</td>
</tr>
<tr>
<td>Chapter 21</td>
<td>-0.22</td>
<td>0.094</td>
</tr>
<tr>
<td>Chapter 22</td>
<td>-0.149</td>
<td>0.114</td>
</tr>
<tr>
<td>Chapter 23</td>
<td>0.044</td>
<td>0.194</td>
</tr>
<tr>
<td>Chapter 24</td>
<td>-0.09</td>
<td>0.194</td>
</tr>
<tr>
<td>Chapter 25</td>
<td>-0.048</td>
<td>0.184</td>
</tr>
<tr>
<td>Chapter 26</td>
<td>-0.035</td>
<td>0.194</td>
</tr>
<tr>
<td>Chapter 27</td>
<td>-0.096</td>
<td>0.224</td>
</tr>
</tbody>
</table>

Table 4. Time offsets from manual timing in seconds on a per-chapter basis.
5.1.4 Five Go Off in a Caravan

5.1.4.1 Manual and Speech Recognition

*Five Go Off in a Caravan* is another medium length book, with a playtime of around 5 hours.

In Figure 11 we demonstrate the accuracy of the whole book, when comparing the manual timings with the speech recognition.

![Errors for the whole book with speech recognition](image)

Figure 10. Sentences offset from 0 in ranges of 0,5 seconds.
In Table 5 we demonstrate the mean and median of each chapter in the book when comparing the manual timings with the speech recognition.

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Mean</th>
<th>Median</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chapter 1:</td>
<td>-1.379</td>
<td>0.07</td>
</tr>
<tr>
<td>Chapter 2</td>
<td>-0.777</td>
<td>-0.182</td>
</tr>
<tr>
<td>Chapter 3</td>
<td>1.43</td>
<td>0.512</td>
</tr>
<tr>
<td>Chapter 4</td>
<td>0.46</td>
<td>0.283</td>
</tr>
<tr>
<td>Chapter 5</td>
<td>0.903</td>
<td>0.339</td>
</tr>
<tr>
<td>Chapter 6</td>
<td>1.915</td>
<td>0.405</td>
</tr>
<tr>
<td>Chapter 7</td>
<td>0.099</td>
<td>0.291</td>
</tr>
<tr>
<td>Chapter 8</td>
<td>0.207</td>
<td>0.3</td>
</tr>
<tr>
<td>Chapter 9</td>
<td>-0.091</td>
<td>0.184</td>
</tr>
<tr>
<td>Chapter 10</td>
<td>1.815</td>
<td>0.282</td>
</tr>
<tr>
<td>Chapter 11</td>
<td>2.801</td>
<td>0.064</td>
</tr>
<tr>
<td>Chapter 12</td>
<td>2.546</td>
<td>0.135</td>
</tr>
<tr>
<td>Chapter 13</td>
<td>-0.252</td>
<td>0.276</td>
</tr>
<tr>
<td>Chapter 14</td>
<td>-0.23</td>
<td>0.257</td>
</tr>
<tr>
<td>Chapter 15</td>
<td>-0.698</td>
<td>-0.183</td>
</tr>
<tr>
<td>Chapter 16</td>
<td>2.395</td>
<td>-0.099</td>
</tr>
<tr>
<td>Chapter 17</td>
<td>0.015</td>
<td>0.297</td>
</tr>
<tr>
<td>Chapter 18</td>
<td>4.425</td>
<td>3.698</td>
</tr>
<tr>
<td>Chapter 19</td>
<td>-0.143</td>
<td>0.215</td>
</tr>
<tr>
<td>Chapter 20</td>
<td>-0.504</td>
<td>0.07</td>
</tr>
<tr>
<td>Chapter 21</td>
<td>-0.029</td>
<td>0.128</td>
</tr>
<tr>
<td>Chapter 22</td>
<td>-0.409</td>
<td>-0.124</td>
</tr>
<tr>
<td>Chapter 23</td>
<td>0.023</td>
<td>0.19</td>
</tr>
<tr>
<td>Chapter 24</td>
<td>-1.379</td>
<td>0.07</td>
</tr>
<tr>
<td>Chapter 25</td>
<td>-0.777</td>
<td>-0.182</td>
</tr>
<tr>
<td>Chapter 26</td>
<td>1.43</td>
<td>0.512</td>
</tr>
<tr>
<td>Chapter 27</td>
<td>0.46</td>
<td>0.283</td>
</tr>
</tbody>
</table>

Table 5. Time offsets from manual timing in seconds on a per-chapter basis.
5.1.5 Eric

5.1.5.1 Manual and Speech Recognition

Eric is the shortest book, with only a playtime of 4 hours.

In Figure 12 we demonstrate the accuracy of the whole book, when comparing the manual timings with the speech recognition.

![Errors for the whole book with speech recognition](image)

Figure 11. Sentences offset from 0 in ranges of 0.5 seconds.

In Table 6 we demonstrate the mean and median of each chapter in the book when comparing the manual timings with the speech recognition.

<table>
<thead>
<tr>
<th>Chapter 1</th>
<th>Mean</th>
<th>Median</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.86</td>
<td>-0.11</td>
<td></td>
</tr>
</tbody>
</table>

Table 6. Time offsets from manual timing in seconds on a per-chapter basis.
5.2 Performance

5.2.1 Time to create a timings file

In Figures 13 we display the time necessary to create all the speech timings files used in the project. You can easily see that chapter structured audio files is much faster than non-chapter structure but it also depends on the size of the book.

In this sample set we can see a clear difference between a chapter structure and a non-chapter structure. For example *Casino Royale* and *Five Go Off in a Caravan* are of comparable length, yet very different in creation time. We can also see that *The Hunger Games* takes the most time to create because it uses a non-chapter structure even if it's about half the length of *The Fellowship of the Ring* and only about a fourth of the length of *A Dance with Dragons* that uses chapter structure.

![Time spent creating a timings file using speech recognition](image)

*Figure 12. Time to create a speech recognition timings file.*
5.2.2 Opening books with differing chapter structure

In this performance test, Figure 14, the evaluation was focused on testing the difference between opening two books of similar size, where one has a simple, flat chapter structure and the other has a parts structure with chapters within them. The blue line is the flat chapter structure with 23 chapters, whereas the red is a book divided into three parts with 28 chapters. The time difference is due to switching folders to open a book.

![Instrumentation Profiling Report](image)

Figure 13. Note the blue shape between 4s and 7s and the red shape between 7s and 10s on the x axis.

5.2.3 Creating a speech recognition timings file with a chapter audio file structure

Figure 15 represents the CPU usage when creating a timings file with a chapter structure. Every chapter starts with a peak in the usage and goes down before the next peak. The long and stable part at the end is the creation of the MD5 checksums for the audio files. During this test the book *Five Go Off in a Caravan* was used.

![Instrumentation Profiling Report](image)

Figure 14. Creating a speech recognition timings file for *Five Go Off in a Caravan*. 

23
5.2.4 Creating a speech recognition timings file with a single audio file

Figure 16 shows the creation of a timings file for a book with a non-chapter structure. The long and stable parts between the two first spikes is when the system progresses the part of the audio file that does not include the chapter. In this test the book *Eric* was used.

![Instrumentation Profiling Report](image)

Figure 15. Creating a speech recognition timings file for *Eric*.

6 Analysis/Discussion

6.1 Construction and Accuracy

When creating the results, some difference was expected. Sentences typically have several hundred milliseconds of silence between sentences and paragraphs, and users have different timing every time when they choose to mark the end of a sentence in the system. This leads to us expecting an offset of about 500ms one way or another. Typically this will veer towards a negative value in comparison, because a user timing will typically wait for a brief respite after the reader has finished speaking. Values can be offset, however, by incorrect values.

*The Hunger Games* is a great example of this. *The Hunger Games* has some discrepancies in Chapter 27, which is due to a few missed sentences on the timer’s end. As the synchronization system works on a line-by-line basis which uses the sentences as a reference, a missed sentence throws that off, so attempts were made to correct it but the results are not perfect. It also had a failure in speech recognition towards the end of Chapter 3, very large misses by the speech recognition engine which pushed the mean up by about 5-6 seconds. This led to the mean offset becoming 0.13 seconds, rather than the -0.5 to -1 that we otherwise expected. In practice, however, this difference is not very noticeable to the end-user, and the system mainly works as expected.

Other books highlighted some weaknesses in our system. *Casino Royale* especially very clearly showed flaws in how we have constructed what a sentence is because of the use of name abbreviations. This could be improved, and changing this would improve results not only for *Casino Royale*, but all books.

Early on in the project we believed it would be best to not treat question marks and exclamation points as sentence ends, because that led to many short sentences. However, this was not based on
empirical data showing us a lack or increase of precision, but rather false assumptions, which we learned was a mistake as we started creating manual timings files. *Casino Royale* is still one of the best examples in the system, with a high rate of accuracy, with only an offset of 340 milliseconds, well within the expected deviation.

*Eric* showcased a new problem in our system in that it had no chapters. The reader is also very expressive, reading quickly, not stopping at punctuation and using many voices. This lead to a low rate of precision in the speech recognition engine. It also had a particularly poor rate of estimation, as it is best when it has short chapters to get a good characters per second value from.

*Five Go Off in a Caravan* is a short book with an ideal book and audio structure. The only issues with this particular book is that it sometimes skips ahead as if someone is fast forwarding a cassette tape, which reduces accuracy. It is therefore one of the lower accuracy books that we have tried. However it showcases the strengths of our estimation algorithm, particularly due to its chapter structure, and creates a high quality timings file. Despite its difficulties the timings file still manages a mean offset of 0.6 seconds from the manual timings file.

Given more time we would have liked to have a wider range of tested books, but creating manual timings files is a time consuming process and we believe that the books we chose are representative of a wide range of both audio quality, writing style and readers.

### 6.2 Performance

The overall system performance is good when using the system for its intended purpose, syncing up between an audiobook and an eBook. The process is instantaneous, and gives the user a sense of the two being tightly connected. Both going from audio to text and having your selection being read has a response time in the milliseconds range.

The computer resource use of the system is not ideal. It uses approximately 70mb RAM for *The Hunger Games* for holding every sentence of the book in its memory. Furthermore there is the added memory restraint of the changed MP3 files. This is a byproduct of the way our system is designed, and to change it would require very significant re-engineering and rethinking.

A bigger problem is the time and resources it takes to create a timings file using speech recognition. As can be seen in figure 8a, it routinely takes about an hour to create a timings file of larger works, using a modern Intel i5 processor on a desktop. It also uses a minimum of 20% of the CPU to process it, with peaks of around 40% when starting new chapters. For a normal user, this would be an inconvenience, but it also doesn’t fit the main use-case for a software such as this. The ideal usage scenario for a consumer facing version of this sort of synchronization software would be mobile platforms, which typically do not have 3GHz+ multi-core processors to create timings files. It would be a massive performance and battery drain on those platforms.

The shortest time we’ve encountered is 5 minutes for *Five Go Off in a Caravan*, which is more acceptable, but still not an ideal situation. That is one of the best case scenarios, as it is a short book which also has, as aforementioned, an ideal file and book structure. *The Hunger Games* is of similar length, but takes an hour instead of five minutes due to being a single, large file.
A performance issue is the amount of time it takes to simply open a book, especially with a timings file related to it. It takes several seconds, and it makes the software feel unresponsive. This could be improved in future revisions using multi-threading to make the system feel more responsive and have tasks running in the background.

7 Conclusions

We constructed a synchronization system that connects audiobooks and eBooks and we’ve tried to be as thorough as possible in testing them. Manually timing files is an inexact science, which creates some deviance, but silences in books after sentences offset this. We found that most of the books we tested were in the expected range of deviation, and are usable with the automatic speech recognition timing system.

The ideal platform for this system would be mobile devices, with timings files being created on desktops or, even better, using cloud/distributed computing.

We created a checksum system for the timings files, which would allow them to be used on different machines, given the correct files. It also ensures that users always have the correct files attached to their eBooks. This could in theory allow for distributed timings files created from company and distributed to users based on their checksums from their audio files.

For future work it would be possible to add highlighting to the text while listening to the audio book, this could for example be great for people that are trying to learn a new language. More multithreading could be added to make it even more fluent for the user and with more threads it can be possible to create timings files even faster.

This would be the best solution for a company which markets both audio and eBooks, such as Barnes and Noble, which want their customers to have the option of using a system such as ours. Alternatively a standard could be implemented for timing audiobooks at the recording, much as it is for creating subtitles, which would allow manually timed files to be present for every book henceforth recorded. This would, unfortunately, severely limit the catalogue of files that can be synced, if used as the only method.
References


   http://www.idpf.org/epub/30/spec/epub30-overview.html

   http://dx.doi.org/10.1108/01435121111158547

   http://dx.doi.org/10.1016/S0098-7913(99)80142-7.

    Euro Conferences

    http://cmusphinx.sourceforge.net/wiki/research/
A. Removal of inaccurate timings

This code example loops through all the sentences in the current chapter and sets suspicious values to zero or a correct value if possible.

```csharp
for (int i = 0; i < this.currentChapter.Sentences.Count; i++) {
    // Step one, find the previous position that has a position and
duration not set to zero.
    AudioPosition previousPosition = new AudioPosition();
    int decrement = 1;
    do {
        // If we're out of range we should break the loop.
        if (i - decrement < 0) {
            if (this.lastChapter != null)
                previousPosition =
this.lastChapter.GetLastSentenceWithValues().FirstAudioPosition;
            break;
        }
        previousPosition =
this.currentChapter.Sentences[i - decrement].FirstAudioPosition;
        decrement++;
    } while (previousPosition.Position == TimeSpan.Zero
        || previousPosition.Duration == TimeSpan.Zero);

    // The second step is to get the closest audio position to the
    // previous position.
    int audioFileIndex =
this.currentChapter.Sentences[i].FirstAudioPosition.AudioFileIndex;
    AudioPosition closestAudio =
this.currentChapter.Sentences[i].GetClosestAudioPosition(
    previousPosition.Position);
    closestAudio.AudioFileIndex = audioFileIndex;

    // The third step is to find the next position that has a position
    // and duration not set to zero.
    AudioPosition nextPosition = new AudioPosition();
    if (i != this.currentChapter.Sentences.Count - 1) {
        int increment = 1;
        bool lastSentence = false;

        // Loop until we find a position that appears to be correct.
        while (true) {
            do {
                // If we're out of range we should break the loop.
                if (i + increment >= this.currentChapter.Sentences.Count)
                {
                    lastSentence = true;
                    break;
                }
            } while (nextPosition.Position == TimeSpan.Zero
                || nextPosition.Duration == TimeSpan.Zero);

            nextPosition =
this.currentChapter.Sentences[i + increment].GetClosestAudioPosition(
    previousPosition.Position);
            increment++;
        }
    }
}
```
// Estimate the interval where the next position should be in.
TimeSpan estimatedShouldBe = TimeSpan.FromSeconds(EstimatePositionInSeconds(i + increment, offset));
TimeSpan estimatedShouldBeMin = estimatedShouldBe - TimeSpan.FromSeconds(90);
TimeSpan estimatedShouldBeMax = estimatedShouldBe + TimeSpan.FromSeconds(90);

// Calculate the interval where the next position should be.
TimeSpan calculatedShouldBe = closestAudio.Position + closestAudio.Duration;
TimeSpan calculatedShouldBeMin = calculatedShouldBe - TimeSpan.FromSeconds(45);
TimeSpan calculatedShouldBeMax = calculatedShouldBe + TimeSpan.FromSeconds(45);

// If the next position is within the estimated or calculated interval // or if we are on the last sentence, we have found a position that // appears to be correct.
if ((nextPosition.Position > estimatedShouldBeMin
    && nextPosition.Position < estimatedShouldBeMax)
    || (calculatedShouldBe != TimeSpan.Zero
        && nextPosition.Position > calculatedShouldBeMin
        && nextPosition.Position < calculatedShouldBeMax)
    || lastSentence) {
    offset = nextPosition.Position - estimatedShouldBe;
    break;
}

// If we're at the last sentence the next position is // calculated instead by using the previous position.
nextPosition = new AudioPosition() {
    Position = previousPosition.Position + previousPosition.Duration + TimeSpan.FromMinutes(1),
    Duration = TimeSpan.Zero
};

// If next position is set to zero, calculate the position.
if (nextPosition.Position == TimeSpan.Zero)
    nextPosition.Position = previousPosition.Position + TimeSpan.FromMinutes(5);

// The final step is to check if the position is not within the previous and next position.
if (closestAudio.Position < previousPosition.Position.Add(previousPosition.Duration)
    || closestAudio.Position > nextPosition.Position) {
    closestAudio.Position = TimeSpan.Zero;
    closestAudio.Duration = TimeSpan.Zero;
}

this.currentChapter.Sentences[i].FirstAudioPosition = closestAudio;
B. Filling missing values

This code example loops through all the sentences in a chapter and calculates missing position values and duration if possible. We estimate the duration if we could not calculate it.

```csharp
for (int i = 1; i < this.currentChapter.Sentences.Count; i++) {
    // If the sentence already has a position or if the previous sentence doesn't have a duration, continue to the next loop.
        || this.currentChapter.Sentences[i - 1].FirstAudioPosition.Duration == TimeSpan.Zero)
        continue;

    // Set the position.
    this.currentChapter.Sentences[i].FirstAudioPosition.Position =
        this.currentChapter.Sentences[i - 1].FirstAudioPosition.Position
        + this.currentChapter.Sentences[i - 1].FirstAudioPosition.Duration;

    // If the next sentence has a position, use it to calculate the duration.
    if (((i + 1) < this.currentChapter.Sentences.Count
        // Calculate the duration. The duration can't be less than zero.
    }
    else
        this.currentChapter.Sentences[i].FirstAudioPosition.Duration =
            TimeSpan.FromSeconds(this.currentChapter.Sentences[i].CharCount / charsPerSecond);

    // If the book uses a non chapter structure, calculate the audiofile index.
    if (this.structure != AudioBookStructure.CHAPTER_STRUCTURE) {
        TimeSpan time = this.currentChapter.Sentences[i].FirstAudioPosition.Duration;
        int audioFileIndex = 0;
        for (audioFileIndex = 0; audioFileIndex < this.mp3Files.Count; audioFileIndex++) {
            time -= this.mp3Files[audioFileIndex].TotalTime;

            if (time < TimeSpan.Zero)
                break;
        }
    }
    this.currentChapter.Sentences[i].FirstAudioPosition.AudioFileIndex = audioFileIndex;
}
```
C. Compare two books and calculates the differences

This code example calculates the differences for every sentence in the two books.

```csharp
/// <summary>
/// Helper for comparing two books.
/// </summary>
/// <param name="a">The baseline book.</param>
/// <param name="b">The book to compare.</param>
/// <returns>The differences.</returns>
private BookDiff CompareBooks(BookDiff a, BookDiff b)
{
    BookDiff value = new BookDiff();
    value.CompletedTime = a_COMPLETEDTime - b_COMPLETEDTime;
    for (int i = 0; i < a.Chapters.Count; i++)
    {
        value.Chapters[i].Add(new ChapterDiff());
        for (int j = 0; j < a.Chapters[i].Sentences.Count; j++)
        {
            value.Chapters[i].Sentences.Add(new AudioPosition());
            value.Chapters[i].Sentences[j] =
                b.Chapters[i].Sentences[j] - a.Chapters[i].Sentences[j];
        }
    }
    return value;
}
```
## D. System Normal Use

![Audiobook Reader Interface](image)

### Table of Contents

- Title Page
- Dedication
- Contents
- Part I: "The Tributes"
  - 1
  - 2
  - 3
  - 4
  - 5
  - 6
  - 7
  - 8
  - 9
- Part II: "The Games"
  - 10
  - 11
  - 12
  - 13
  - 14
  - 15
  - 16
  - 17
  - 18
- Part III: "The Victor"
  - 19
  - 20
  - 21
E. Manual Timing

"I thought we were going to go back to my room," Eric complained.

"I thought that, too," said Rincewind, shutting to be heard over the rumbling.

"Snap your fingers again, demon!"

"Not on your life! There's plenty of places worse than this!"

"But it's all hot and dark!"

Rincewind had to concede that it was also shaking and noisy. When his eyes grew used to the blackness he could make out a few spots of light here and there, whose dim radiance suggested that they were inside something like a boat. There was a definite feel of carpentry about everything, and a powerful smell of wood shavings and glue. If it was a boat, then it was having an awfully painful launching down a slipway greased with rock.

A jolt shung him heavily against a bulkhead.

"I must say," complained Eric, "if this is where the most beautiful woman in the world lives I don't think much of her choice of boatwork. You'd think she'd put a few cushions or something around the place!"

"Boodah!" said Rincewind.

"She's bound to have one," said Eric snuffily. "I've read about 'em. She reclines on 'em!"

"Tell me," said Rincewind, "have you ever felt the need to have a cold bath and a brisk run around the playing fields?"

"Never!"

"It could be worth a try!"

The world stepped along...
F. Comparison window

![Comparison window]

- Base Timing File: ...
- Timing File: ...

[Compare] [Save] Status: select files
G. Complete flowchart for opening a book

1. Start
2. Prompt the user to select an EPUB file
3. Parse e-book
4. Load mp3 files from the timings file
5. Unload all loaded mp3 files

   - Timings file exists?
     - Yes: Load mp3 files from the timings file
     - No: Open selected ebook in new tab

   - Progress file exists?
     - Yes: Load progress file
     - No: Prompt the user to select mp3 files

   - Load progress file
     - Yes: Load progress file
     - No: Prompt the user to select mp3 files

   - Load selected mp3 files
     - Yes: Unload all loaded mp3 files
     - No: Set last known volume and position

   - Prompt the user to select an EPUB file
     - Yes: Parse e-book
     - No: Open selected ebook in new tab

   - Progress file exists?
     - Yes: Load progress file
     - No: Prompt the user to select mp3 files

   - Load progress file
     - Yes: Load progress file
     - No: Prompt the user to select mp3 files

   - Load selected mp3 files
     - Yes: Unload all loaded mp3 files
     - No: Set last known volume and position

   - Prompt the user to select an EPUB file
     - Yes: Parse e-book
     - No: Open selected ebook in new tab

   - Prompt the user to decide if the progress file should be deleted
     - Yes: Delete progress file
     - No: Unload all loaded mp3 files

   - Delete progress file
     - Yes: Delete progress file
     - No: Unload all loaded mp3 files

   - Prompt the user to decide if the progress file should be deleted
     - Yes: Delete progress file
     - No: Unload all loaded mp3 files
H. Complete source code

The complete source code for the example program can be found here
http://git.io/bWMFHg