

Accessible EPUB

Bachelor Thesis
by

Sachin Rajgopal

Study Centre for the Visually Impaired Students (SZS)
Department of Informatics

First Reviewer:
Second Reviewer:
Supervisor:

Prof. Dr. Rainer Stiefelhagen
TODO: Eintragen
Dr. Thorsten Schwarz

Project Period: 01/12/2017 – 30/03/2018

Ich versichere wahrheitsgemäß, die Arbeit selbstständig angefertigt, alle benutzten Hilfsmittel vollständig und genau angegeben und alles kenntlich gemacht zu haben, was aus Arbeiten anderer unverändert oder mit Abänderungen entnommen wurde.

Karlsruhe, den 29.03.2018

.....
(Sachin Rajgopal)

Zusammenfassung

TODO: Zusammenfassung (Deutsch)

Abstract

Current document standards have the characteristic that they can only serve one group of users under the aspect of vision (sighted, visually impaired or blind reader). The logical conclusion of this would be that a document standard which can serve all three groups would be better for accessibility. This thesis presents a new approach of an universal accessible version of EPUB 3 documents, which will allow sighted, visually impaired and blind readers to use and share the same EPUB 3 document by an "integrated switching mechanism" to change the output format and how the document is displayed. The new document standard will be tested on several reading systems to examine how well it works on them. Furthermore, a simple editor, which will be similar to existing word processors, will be presented that allows users to easily create accessible EPUBs of the new document standard without knowing how an EPUB file is constructed or created.

Contents

1	Introduction	1
1.1	Motivation	1
1.2	EPUB	2
1.2.1	Changes from EPUB 2 to EPUB 3 and EPUB 3.1	3
1.2.2	EPUB Structure	4
1.2.2.1	Package document	4
1.2.2.2	Navigation	6
1.2.2.3	Content documents	6
1.3	Goal of this thesis	7
1.3.0.1	Document standard	7
1.3.0.2	Editor	8
2	Related Work	9
3	EPUB Document Standard	13
3.1	Requirements	13
3.1.1	Switching between versions	13
3.1.1.1	JavaScript	13
3.1.1.2	CSS	14
3.1.2	Text	16
3.1.3	Images	16
3.1.4	Mathematical formulas	18
4	Evaluation EPUB Standard	23
4.1	Reading systems on Android	24
4.1.1	Tolino	24
4.1.2	Reasily	25
4.1.3	Gitden Reader	26
4.1.4	FBReader	26
4.1.5	UB Reader	27
4.2	Reading systems on iOS	27
4.2.1	iBooks	27
4.3	Reading systems on Windows	28
4.3.1	Calibre	28
4.3.2	Readium	29
4.3.3	Bluefire Reader	30
4.3.4	Icecream Ebook Reader	31
4.3.5	AZARDI	31
4.4	Discussion: what do developers have to do?	32

5	AccessibleEPUB Editor	33
5.1	Requirements	33
5.2	Programming language	33
5.3	Features	34
5.3.1	Main window with editor and preview	34
5.3.1.1	Editor	34
5.3.1.2	Preview browser	35
5.3.2	Inserting mathematical equations	37
5.3.3	EPUB work behind the scenes	40
5.3.3.1	Inserting images tables	41
6	Future Work	45
7	Conclusion	47
	Bibliography	49

1. Introduction

1.1 Motivation

In recent years, accessibility has become increasingly important, especially for accessible documents. Several states have passed regulatory laws that ensure equal treatment of all people and ensure that documents are accessible to all [18].

The currently dominant format for accessible electronic documents are Microsoft Word and PDF (Portable Document Format) documents, or more precisely PDF/UA (PDF/Universal Accessibility) documents [1]. First of all, both formats have a predefined page size. While this is useful for printed documents, a computer screen can rarely display all contents of the document to the detriment of visually impaired people [15], as shown in the example of figure 1.1, where the columns are only as long as part of the screen. Therefore, an electronic document format without a fixed document size containing semantic and structural information and a fixed reading order would be better suited to meet the requirements of accessibility.

Furthermore, different "selectable" forms of presentation would be advantageous, especially for graphics or mathematical formulas. For example, formulas in the \LaTeX source code for blind users or high-contrast images for users with limited residual vision. This could be combined with EPUB 3 [15].

A short overview of the major changes which will be expanded upon in later chapters include [11, 13]:

- Support for HTML5
- MathML
- Using a XHTML document for navigation instead of NCX
- Scripting
- CSS 3 support

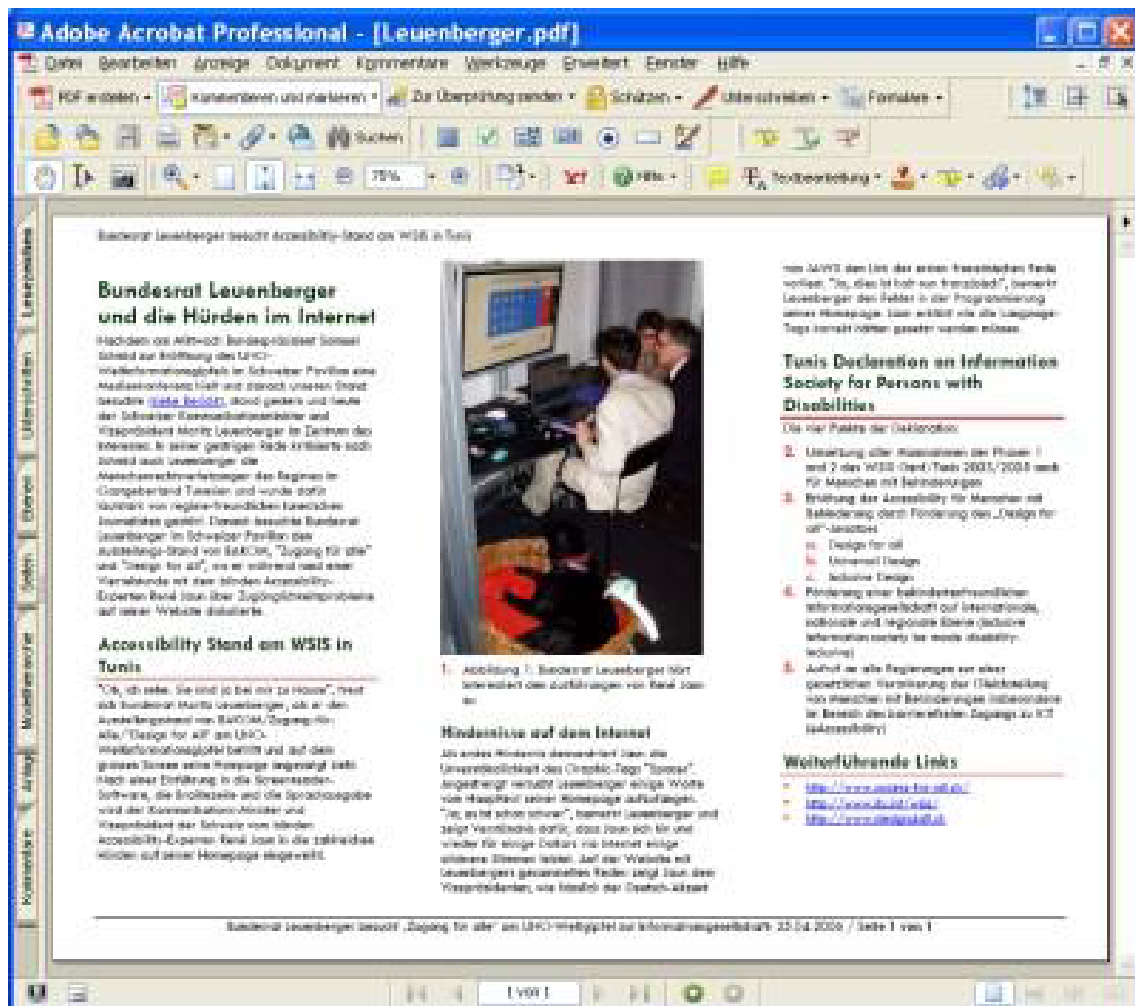


Figure 1.1: Example of a PDF not suitable for the visually impaired [19]

1.2 EPUB

EPUB stands for **E**lectronic **P**ublication and is a format primarily used for books in an electronic format (E-book). The EPUB format was created by the International Digital Publishing Forum (IDPF) and the current version is 3.1 which is a minor update to EPUB 3¹ [12]. EPUB uses XML based formats like XHTML, and thus also uses the accessibility standards and guidelines already established in many nations like the Web Content Accessibility Guidelines (WCAG) [22]. This was done as reading systems can have different screen sizes and the EPUB content can therefore be reflowable. Font type and size can also be adapted to the individual needs of the users. Visually impaired people could therefore adjust the document to their preferences in font style, size and color. The EPUB 3 specification also contains guidelines for accessibility so these features are built in and not an afterthought [8]. For example while EPUB document have reflowable content, they still support page numbers so that printed book users can refer to a page number and the EPUB user can find it.

¹<http://www.idpf.org/epub/31/spec/epub-changes.html>

Name	Änderungsdatum	Typ	Größe
META-INF	08.01.2018 17:51	Dateiordner	
OEBPS	08.01.2018 17:51	Dateiordner	
mimetype	08.01.2018 17:51	Datei	1 KB

Figure 1.2: Folder structure within an EPUB file

1.2.1 Changes from EPUB 2 to EPUB 3 and EPUB 3.1

The EPUB working group has also made some important changes from EPUB 2 to EPUB 3 to make it more accessible. Equations can now be displayed in MathML [11]. In EPUB 2 they had to either be displayed in normal XHTML or as images. It is difficult to write math in normal XHTML, as complex symbols like fractions, roots and plus-minus might not be displayed properly. Images can display the equations properly, but only if they are SVG files can be scaled without quality loss. Furthermore, all images are not as accessible as MathML, which can be scaled properly and can be read by a screen reader.

In EPUB 2 the navigation was done with a Navigation Control file for XML (NCX) file, usually named `toc.ncx` [8]. A NCX file is also written in XML, but the tags are different from XHTML. In EPUB 3 the navigation document is now done instead with a XHTML file, and can be created with regular XHTML elements like `ol` and `ul`, for ordered and unordered lists, respectively. XHTML is much more widely used and is already an important format in EPUB 3, while NCX files have a distinct syntax, like `<navMap>` for tables, which has to be learned in addition to XHTML. Replacing NCX means that the creator has one less document format to worry about.

Scripting with JavaScript was strongly discouraged in EPUB 2, but in EPUB 3 it is optional [8]. This allows documents to be interactive, but it is recommended that information is not hidden if a device does not support scripting. This should be done with a process called progressive enhancement. In essence, content should first be designed in the most accessible way, and then layers of enhancement should be added, which might not be supported on all devices. For example, if the creator wants to add an animation with JavaScript to the document, it might be better to add several images showing the stills of the animation at important junctions. Each still image would also have alternative text for users with screen readers. The creator could then add an animation for devices that support it, enhancing their experience.

EPUB 2 supports Cascading Style Sheets (CSS) 2, while EPUB 3 supports CSS 2.1 with added modules from CSS 3 which were defined as a basic EPUB CSS Profile. The EPUB CSS Profile was removed in EPUB 3.1 and the IDPF defined more general CSS support requirements in its place [13]. Henceforth they also use the more general definition of CSS as mentioned by CSS Working Group, which results in creators not being required to learn specific CSS attributes only seen in EPUB.

One new feature in EPUB 3 are media overlays, which might be of considerable interest to people with accessibility requirements [8]. This feature is supposed to help to integrate Digital Accessible Information System (DAISY) (digital talking books) in EPUB 3. The first DAISY Standard originated in Sweden in 1994. DAISY is a digital standard to

```
application/epub+zip
```

Figure 1.3: Contents of the `mimetype` file

```
<?xml version="1.0" encoding="UTF-8"?>
<container xmlns="urn:oasis:names:tc:opendocument:xmlns:container" version
="1.0">
<rootfiles>
<rootfile full-path="EPUB/package.opf" media-type="application/oebps-
package+xml"/>
</rootfiles>
</container>
```

Figure 1.4: Contents of `container.xml`

create an audio substitute for printed media [7]. When it was created the primary audio media were tapes and CDs, both with rather limited runtime. Discs with DAISY books, however, can hold up to 40 hours of audio, while the corresponding number for CDs is about 74 minutes [3]. DAISY books also allow the user to skip between chapters, pages or even sentences and create bookmarks. EPUB 3 has the same features, but media overlays now allow it to synchronize audio narration to text. The user can still use Text-To-Speech(TTS) rendering, but as it is computer generated, it might not pronounce all words properly. Prerecorded audio narration is of course better at this, and now the user can switch between reading and audio narration without navigating in an audio file.

However, many of these changes are not supported by reading devices, and this will be the topic of discussion in chapter 4, where reading devices are evaluated, both software and hardware [11].

1.2.2 EPUB Structure

What is EPUB exactly? It has the extension `.epub`, but is actually just a renamed ZIP file(extension `.zip`). After changing the file extension, the new ZIP file can be decompressed and the files within the document can be accessed [8].

Compressing the folder back to a ZIP file has to be done with care. The `mimetype` should be the first file to added to the ZIP container and not the other two folders. If this is not done properly, the EPUB file is not in the proper format and some readers can not read the file.

The folder structure of an EPUB file is shown in figure 1.2. The file named `mimetype` has a single line indicating that the ZIP file contains an EPUB. The META-INF folder contains just one file named `container.xml`. This file has to indicate the location of the package file (extension `.opf`). The name of the folder named OEBPS can actually be freely chosen, however, `container.xml` has to describe the path to the package file correctly. OEBPS stands for Open eBook Publication Structure, which is the format superseded by EPUB [14].

1.2.2.1 Package document

The OEBPS folder contains the package document which has the extension `.opf`. The package document contains five XML declarations:

```

<metadata xmlns:dc="http://purl.org/dc/elements/1.1/" xmlns:opf="http://www
.idpf.org/2007/opf" xmlns:dcterms="http://purl.org/dc/terms/">
  <dc:title>BookTitle</dc:title>
  <dc:creator>Author</dc:creator>
  <dc:language>en</dc:language>
  <dc:publisher>Publisher</dc:publisher>
  <dc:identifier id="BookId">BookIdentificationNumber</dc:identifier>
  <meta property="dcterms:modified">2018-02-07T10:43:33Z</meta>
  <meta name="AccessibleEPUB" content="0.1.0" />
</metadata>

```

Figure 1.5: The various Dublin Core tags in the `metadata`

```

<manifest>
  <item id="ncx" href="toc.ncx" media-type="application/x-dtbnx+xml" />
  <item id="Content.xhtml" href="Text/Content.xhtml" media-type="application/xhtml+xml" />
  <item id="style.css" href="Styles/style.css" media-type="text/css" />
  <item id="navid" href="Text/nav.xhtml" media-type="application/xhtml+xml" properties="nav" />
</manifest>

```

Figure 1.6: Files listed in the `manifest`

- `metadata` (required)

The `metadata`, which refers to data that provides information about other data², declaration information about the document in Dublin Core form [16]. The Dublin Core Metadata Initiative is an organization managing terms used in metadata documents. Only `dc: title`, `dc:language`, `dc: identifier` and one `meta` are required, but there are several optional ones³. The data entries have to follow the specifications.

- `manifest` (required)

The `manifest` declarations contains all of the files which should be included in the EPUB file with an ID, the path to each file and an appropriate declaration of the media type. Properties also have to entered. If a XHTML file is a navigation document, `nav` has to be entered in the properties.

- `spine` (required)

The `spine` declares the default reading order of the EPUB. The reader is free to go to whatever page they want, but the `spine` allows the document creator to set a logical order of the files in the spine. The only file types which do not require a fallback in the `spine` are XHTML and SVG files [8].

- `guide` (optional)

²<https://www.merriam-webster.com/dictionary/metadata>

³<http://dublincore.org/documents/dcmi-terms/>


```
<spine toc="ncx">
  <itemref idref="Content.xhtml"/>
</spine>
```

Figure 1.7: The reading order declared in the `spine`

```
<nav epub:type="toc" id="toc">
<h1>Table of Contents</h1>
  <ol>
    <li>
      <a href="../Text/Content.xhtml">Start</a>
    </li>
  </ol>
</nav>
```

Figure 1.8: An example table of contents

- `bindings`. (optional)

The `guide` is deprecated and only needed for EPUB 2, but it is useful for backwards compatibility. `bindings` is used for fallback options for interactive media. Both of these are optional and will therefore not be discussed further in this bachelor thesis.

1.2.2.2 Navigation

The OEBPS folder also contains a file, normally named `nav.xhtml`, which is the table of contents. The table of contents can be used to navigate to any tagged section and is created by simply making an ordered or unordered list in HTML. A short example is shown in figure 1.8.

1.2.2.3 Content documents

One of the primary documents of an EPUB document are Extensible Hypertext Markup Language (XHTML) files. XHTML is used, because it is a web format and websites have to be displayed on a variety of devices, from phones with screen dimensions of a few inches to 50 inch television screens with varying display resolutions. The content is therefore reflowable and suits Ebook readers as they are available in various sizes.

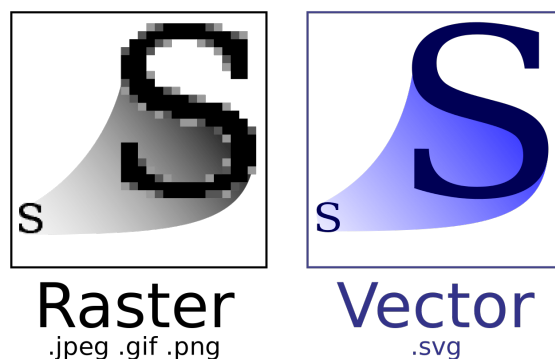


Figure 1.9: Source : https://upload.wikimedia.org/wikipedia/commons/thumb/6/6b/Bitmap_VS_VG.svg/1280px-Bitmap_VS_VG.svg.png

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Figure 1.10: Quadratic Equation

```
{x = \frac{ - b \pm \sqrt{ b^2 - 4ac }}{2a}}
```

Figure 1.11: Quadratic Equation in LaTeX code

Another primary document of EPUB documents are Scalable Vector Graphics (SVG) files. They are based on XML and while image formats like JPEG and PNG are raster based with pixels, SVG files are vector based. This means that they retain their appearance at any size and do not get blurry at larger sizes than they created at, which can be seen in figure 1.9.

1.3 Goal of this thesis

The goals of this thesis are:

1. Create an accessible document standard based on EPUB 3 with the following features:
 - (a) Have one document for all users: sighted, visually impaired and blind users
 - (b) Ability to change between display styles for each group of users
 - (c) Support for MathML
 - (d) The font type changes depending the user group.
2. Develop an editor which will then be able to create EPUB files of this new standard. The main features will include:
 - (a) Creating documents of the new accessible EPUB standard
 - (b) Not requiring programming knowledge
 - (c) Inserting formulas using MathML using only LaTeX math terms
 - (d) Preview the display styles of each target group.

1.3.0.1 Document standard

Existing literature, both electronic and in paper form, is frequently not accessible due to the figures and mathematical formulas in them. If the figure does not have proper captions or alternative text, blind people will be unable to extract the knowledge sighted people will from it. Blind people read formulas with specialized mathematical braille notation, such as Nemeth code in the USA and Marburger mathematical code in Germany [4]. However, few books are printed in braille, braille printers are uncommon and expensive and the resulting books tend to be very heavy. When the textbooks are in electronic form, the mathematical equations have to be written in LaTeX code. The quadratic equation is shown in figure 1.10, while the corresponding LaTeX code is shown in figure 1.11.

Visually impaired people have different requirement for mathematics and graphics. They should scale and not get pixelated when the size is increased, as shown in figure This is best done with SVGs, which were mentioned in last chapter. Visually impaired people also need the text to be a sans-serif font, like Helvetica or Arial [10].

Sighted people have their formulas appear as a serif font as shown in figure 1.10, visually impaired people need it to appear as sans-serif and blind people needs LaTeX code [4]. This is difficult to do in one document, so separate documents have to be created for the blind and the visually impaired, creating additional work for the document creator. The new standard must have all include versions for sighted, visually impaired and blind people in one document to simplify the creation process. Instead of using the currently dominant formats, PDF and Microsoft Word, this will be done with EPUB 3, because they are more suited to electronic formats and allow dynamic content, thanks to CSS 3 and JavaScript. There will be a easy way to switch between three versions. For example, when the blind version is chosen, content such as formulas will change to LaTeX code.

1.3.0.2 Editor

The second goal of this thesis is to develop an editor to create documents of this standard easily. It should not require programming ability and allow insertion of formulas and figures which will be available in all three versions. The editor should not be difficult to learn and be What-You-See-Is-What-You-Get(WYSIWYG), much like a word processor, instead of XHTML which requires coding. It is intended to enable a guided creation of the documents.

2. Related Work

A project which also uses HTML/XHTML to make documents accessible is discussed in a paper [20] by Voegler et al. HTML is chosen as a book format, because it is operating system independent, accessible to screen readers and the content can be adapted visually to a person's wishes. However, the work is not done by original document author, usually university staff, but by students who transcribe it to accessible versions. They have to request the content from authors and get permission. Coding with HTML creates problems like linking errors, invalid HTML-code, etc, so Voegler et al recommend using Markdown [9], which is easier to write than HTML. It can then be converted to a variety of formats using pandoc [17], which also converts LaTeX code to MathML. While reducing the number of mistakes compared to HTML, Markdown still requires programming and only people with sufficient knowledge of it will be able to use it effectively. Students doing the transcription will be more knowledgeable in coding than teachers in schools for the blind.

If EPUB is compared to HTML as book format, it is of course very similar as EPUB uses HTML/XHTML. HTML as a book format can be described as a simplified version of EPUB only without the packaging such as the package document, in essence only having the OEBPS folder. However, images have to be added manually to the folders, and CSS files have to be created. These are extra steps which might result in frequent mistakes. Furthermore, Pandoc is command line based and is therefore more difficult to use than programs with a graphical user interface(GUI). It also has to be run separately after coding in Markdown, instead of being integrated in the creation process. Nevertheless, this paper shows that HTML, if created properly, is suitable to the needs of blind and visually impaired students.

Another approach is offered by Leporini et al in their project Book4All [5]. It originally allowed PDFs to be exported to XHTML or DAISY, later EPUB 3 was added as an option. It reads the PDF with the PDF Viewer to gather information about fonts and recognize if the document is in a multi-column layout. Then Book4All has to take this information and extracts the text and semantic information. This includes keeping the format of the PDF, such as recognizing headings, images and tables being properly tagged. The user has the option of correcting tagging mistakes. It then has to be exported to its final format.

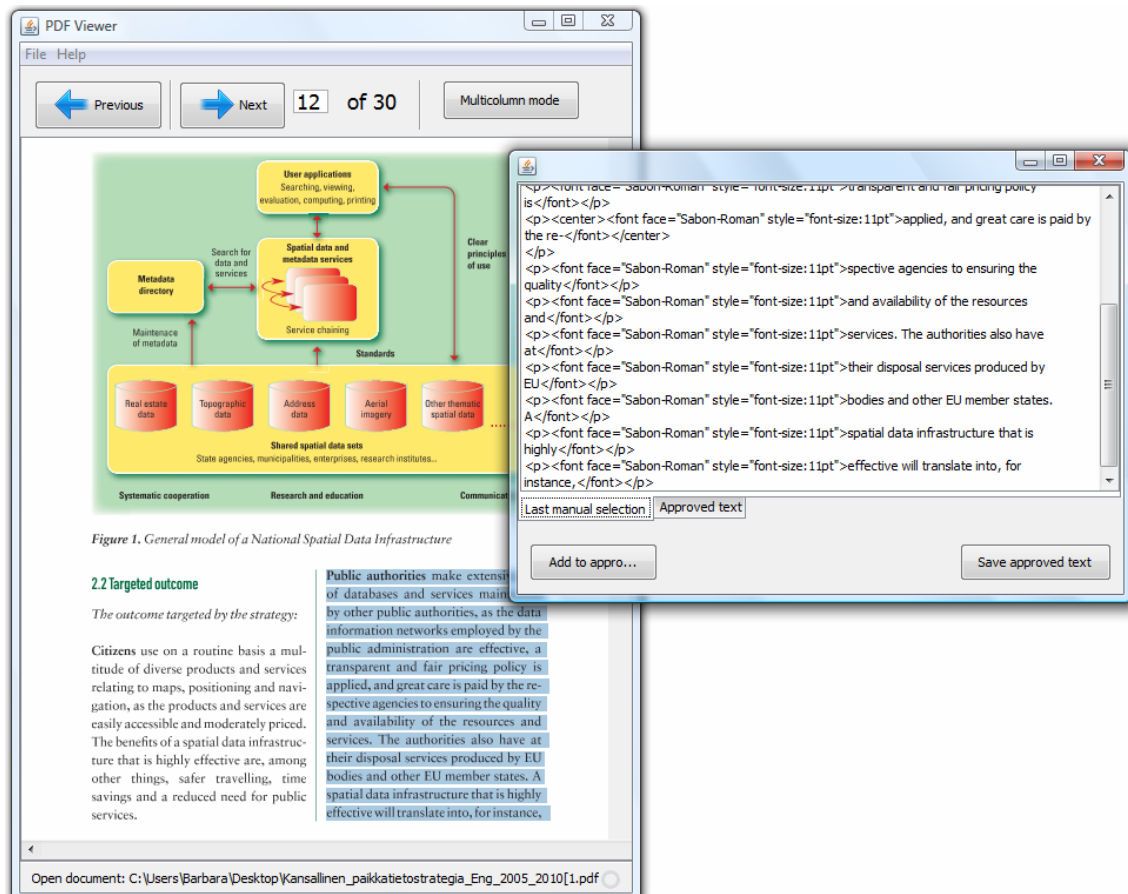


Figure 2.1: A view of the ad-hoc PDF Viewer extracting text from a multi-column layout document
[5]

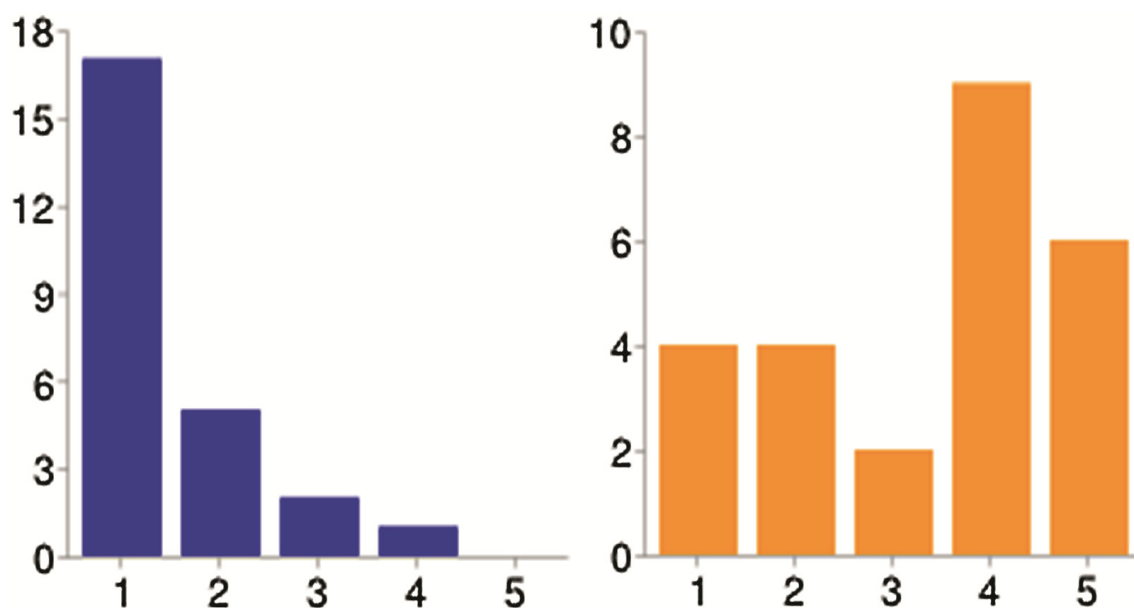


Figure 2.2: Difficulty degree to access the content for the EPUB (blue, left) and PDF (orange, right), respectively (1-not difficult to 5-very difficult) [2]

Book4All attempts to semi-automate the conversion process as far as possible, but still offers the option of post processing the resulting markup as shown in figure 2.1. While this is optional, important parts like alternative text for images can only be added like this. The markup is in Intermediate Book Format (IBF), which is based on XML. Without basic knowledge of XML, post-processing becomes much more difficult.

Bartalesi and Leporini [2] also carried out an online survey in a follow-up work and asked 25 users to rate "enriched" EPUBs created in Book4All in comparison to the original PDF format regarding accessibility and usability. 50% of respondents preferred EPUB over other e-book formats, while 13% said EPUB was equivalent. The sample group also felt that it was easier to access content in EPUBs and use the table of contents than in PDFs. Furthermore, 80% of blind users were unable to read images in PDFs correctly with their screen reader, while the corresponding value for EPUBs was less than 50%. 64% of users found the EPUB's document structure easy to understand. Users also reported that it is much easier to access content in the EPUB than in a PDF, as shown in figure 2.2. The results show that EPUB is suitable format for blind people, if the EPUB contains proper tagging. Those works motivate us to develop a new universal approach.

3. EPUB Document Standard

3.1 Requirements

One requirement of the new standard is that incorporates a way to switch between three versions, them being:

- Blind
- Visually impaired
- Normal/sighted

Switching between three versions should be simple and visible. It should not be hidden behind menus and settings and must be part of the document itself. A document of the standard must conform to EPUB 3 and not rely on external programs to switch between the versions.

The specifications of each version will be compared with each important component of a document.

3.1.1 Switching between versions

Initially the intent was to find one new standard so that EPUB documents can be accessible. Unfortunately, creating a single standard was not possible due to not all EPUB readers supporting all EPUB 3 features.

3.1.1.1 JavaScript

JavaScript is used to make websites interactive, so it was the logical option to dynamically change the versions. The coding was also quite simple and only involved replacing a single line in the CSS file with programming. This was easy to do in JavaScript, and testing the individual XHTML files in the EPUB showed that it was successful.

Shown in 3.1 is style.css which just imports the rules of a CSS file. The methods `selectVisible ()`, `selectImpaired ()` and `selectBlind ()` in script.js all function in the same way. To


```
@import url("../Styles/visible.css");
```

Figure 3.1: Contents of style.css

```
<head>
<title></title>
<link rel="stylesheet" href="../Styles/style.css"/>
<script src="../Misc/script.js"></script>
</head>

<body epub:type="frontmatter" onload="storageCSS();">
<a id="a1" href="#a1" onclick="selectVisible();">Normal</a>
<a id="a2" href="#a2" onclick="selectImpaired();">Visually impaired</a>
<a id="a3" href="#a3" onclick="selectBlind();">Blind</a>
</body>
```

Figure 3.2: The links in the JavaScript version

remember the version chosen, local and session storage is used. First they check if local storage is available. If it is then the method attempts to get the item `accessibleEPUBcurrentCSS`. If it does not exist, then it is created. This item is then set with the same import line in figure 3.1, only with different CSS file name for each version. If local storage is not available the same process is repeated with session storage. Local storage is preferable to session storage as changes are kept even after the program has been exited. This is not the case with session storage so the user has to set version again at the start of the program. Afterwards the method `loadCSS()` is called which deletes the CSS rule of a XHTML file and adds the rule set by the earlier methods.

The content of the XHTML file responsible for the switching mechanism, `Version-Changer.xhtml`, is shown in figure 3.2. It is important that both `style.css` and `script.js` are referred to, or the switch mechanism would be unsuccessful. In the body of the XHTML document, the method `storageCSS()` is called whenever the XHTML file is loaded, so whenever the XHTML are switched back to it from another. This method checks if local and then session storage are available. If either one is available then it sets the item in the storage and calls `loadCSS()`.

There are some issues with JavaScript version changer. If the software or hardware reader does not support temporary storage, the EPUB is always displayed in its default appearance. Due to this, switching with JavaScript is not a reliable option. As mentioned in chapter 1, JavaScript does not have to be supported by all EPUB readers. Nevertheless, the JavaScript implementation supports multiple XHTML files, because temporary storage is supported. For the same reason table of contents are also supported, as they are normally displayed in a separate file, named `nav.xhtml`.

3.1.1.2 CSS

The second method does not use JavaScript and uses some advanced features of CSS, introduced in CSS 3. This mainly refers to CSS selectors. Selectors allow HTML documents to have limited interactive capabilities, such as changing the appearance of a clicked element¹ [23].

Before showing how the selectors work, the format of the content file should be discussed. It is shown in figure 3.3. Unlike the JavaScript version, the links are in the same XHTML

¹https://www.w3schools.com/cssref/css_selectors.asp

```

<body>
  <a class="versionChanger" id="a1" href="#visible">Normal</a>
  <a class="versionChanger" id="a2" href="#impaired">Visually impaired</a>
  <a class="versionChanger" id="a3" href="#blind">Blind</a>

  <div style="padding:none" id="impaired" class="impaired">
    ...
  </div>
  <div id="blind" class="blind">
    ...
  </div>
  <div id="visible" class="visible">
    ...
  </div>
</body>

```

Figure 3.3: The links and divs in the CSS version

```

.visible {
  display:inline;
}

.impaired:target ~ .visible {
  display:none;
}

.blind:target ~ .visible {
  display:none;
}

.impaired {
  display:none;
}

.blind {
  display:none;
}

.impaired:target{
  display:inline;
}

.blind:target{
  display:inline;
}

```

Figure 3.4: The CSS selectors responsible for the dynamic version switching

file and are shown at the beginning of the EPUB file. In the JavaScript version, the links are on a separate page.

The code shown in figure 3.4 is responsible for the switching mechanism. At first only the visible version can be seen and the other two are hidden. The versions appear with the `:target` selector, which affects the appearance of the element if it was a target of a link (`<a> element`). If another link was clicked, then the current version becomes hidden again. However, the visible version would not become hidden, and this was a major problem. CSS selectors have limited capabilities compared to JavaScript, because they were intended to complement and not replace each other. After a bit of experimentation with various CSS selectors, the `~` selector delivered the desired results. `~` allows every element on the right hand side to be selected if it preceded by a left hand side element. So in the case of `.impaired:target ~ .visible`, once the impaired link has been clicked, the visible sections becomes hidden.



Figure 3.5: Comparison of serif(left) and sans serif(right) fonts

Source: <https://cdncms.fonts.net/images/6bff0c2cdbbcca14/A.SerifSansPrint.jpg>

Lorem Ipsum

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Quisque vel placerat felis, ut scelerisque lectus. Integer tristique ipsum a rutrum molestie. Curabitur ultricies lectus auctor, ultrices arcu a, facilisis risus. Nullam in sem orci. Duis porta pulvinar lectus, vitae ultrices nunc lacinia at. Donec sed fermentum ligula. Praesent ligula est, dignissim ut facilisis tempus, pellentesque auctor dui. Integer rhoncus tristique arcu nec facilisis. Suspendisse id metus in leo dictum sollicitudin. Praesent consectetur nunc eget lacus euismod, non accumsan erat condimentum. Sed imperdiet pellentesque iaculis. Vivamus ullamcorper rutrum venenatis. Proin commodo leo vitae metus consectetur, consequat varius ligula condimentum. Pellentesque sed faucibus massa. Sed non nisi at ante congue vulputate.

Figure 3.6: Text of the visually impaired version

3.1.2 Text

The blind version does not have to have any special requirements for text, and its appearance will be identical to the normal version. The text size could be made much smaller, but leaving it at a standard size will allow other people to also read the text, such as for proofreading of the blind version.

The visually impaired version needs to use sans-serif fonts, as seen in figure 3.5, instead of serif fonts, as it easier to read and identify individual characters [10]. Furthermore, the font should be larger than the normal version.

The font of the normal version can be any font, but for this standard a serif font was chosen. Most importantly, the text should not go beyond the edge of the screen so that horizontal scrolling is not required. All normal text in the document will be between `<p>` and `</p>` tags. A comparison of the visually impaired and normal version can be seen in figures 3.6 and 3.7.

3.1.3 Images

All images need to have alternative text to describe them and preferably have a title given to them. This will help blind users as the screen reader will read out both of them.

Lorem Ipsum

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Quisque vel placerat felis, ut scelerisque lectus. Integer tristique ipsum a rutrum molestie. Curabitur ultricies lectus auctor, ultrices arcu a, facilisis risus. Nullam in sem orci. Duis porta pulvinar lectus, vitae ultrices nunc lacinia at. Donec sed fermentum ligula. Praesent ligula est, dignissim ut facilisis tempus, pellentesque auctor dui. Integer rhoncus tristique arcu nec facilisis. Suspendisse id metus in leo dictum sollicitudin. Praesent consectetur nunc eget lacus euismod, non accumsan erat condimentum. Sed imperdiet pellentesque iaculis. Vivamus ullamcorper rutrum venenatis. Proin commodo leo vitae metus consectetur, consequat varius ligula condimentum. Pellentesque sed faucibus massa. Sed non nisi at ante congue vulputate.

Figure 3.7: Text of the normal version

Usually the alternative text will be set as a property and the screen reader will have to identify the image. However, there was an issue. In some programs the screen reader does not read out the alternative text, so a different way had to be found to display it. The image will be part of a figure element of HTML5, shown in figure 3.10. A figure can also contain a caption, which will be identified by the screen reader as such, seen in figure 3.9 Furthermore, the alternative text will be inserted as a paragraph with the `<p>` element. It will remain invisible in normal and visually impaired mode, but will appear as normal text in blind mode. The text should be surrounded by specific tags, like `<Image>` or `<Graph>` shown in figure 3.8. It also has the CSS class "transparent", which only appears in blind mode.

Images in visually impaired and normal mode will be displayed normally, but the image, caption and figure will all have a maximum length of 100% of the screen size.

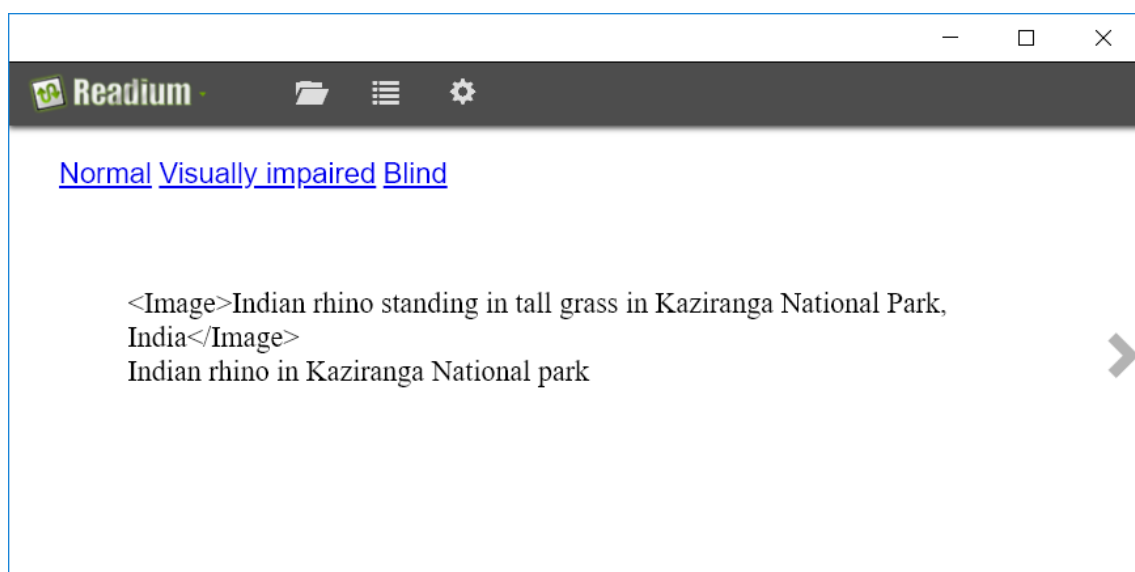


Figure 3.8: Image in 'Blind' mode

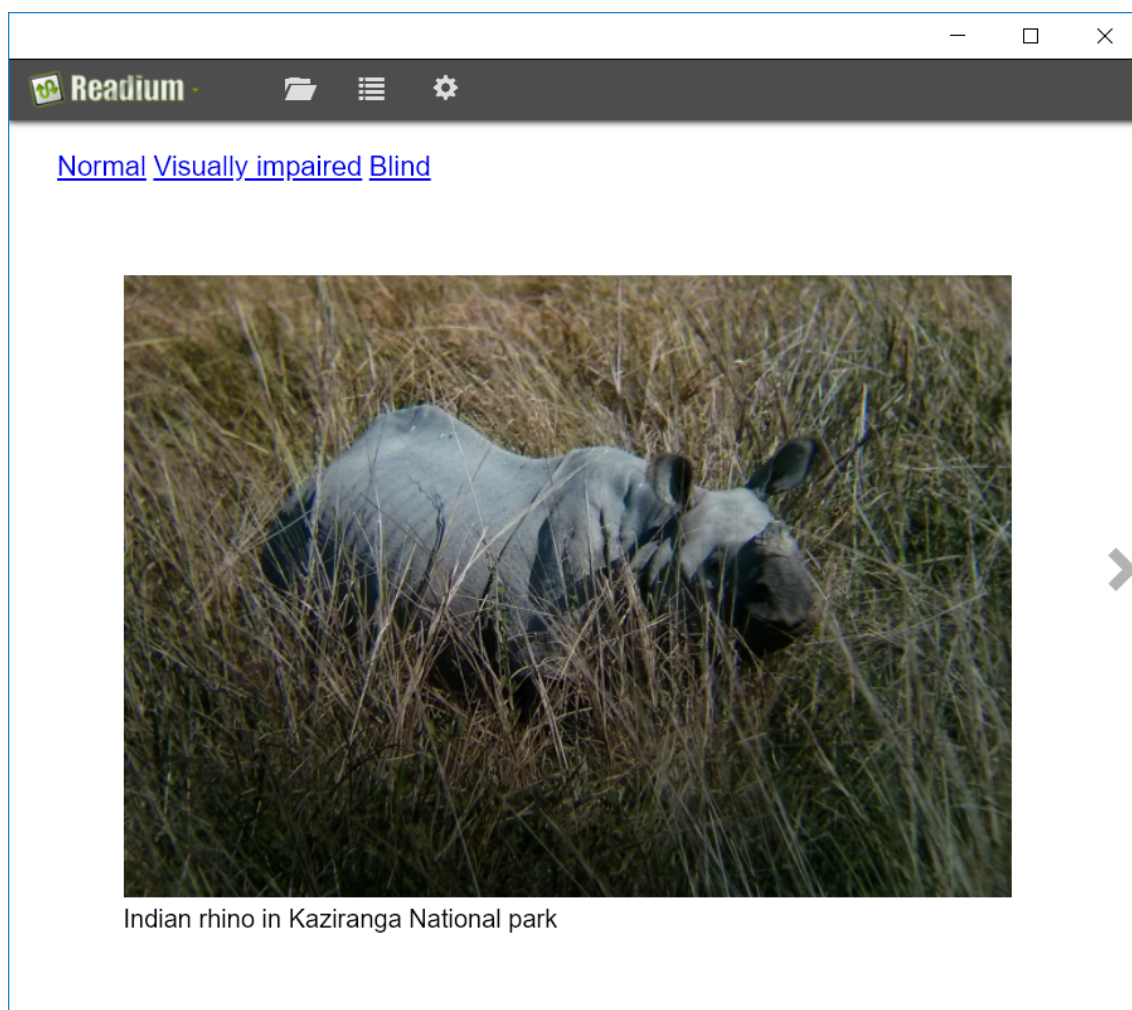


Figure 3.9: Image in 'Visual impairment' mode

```

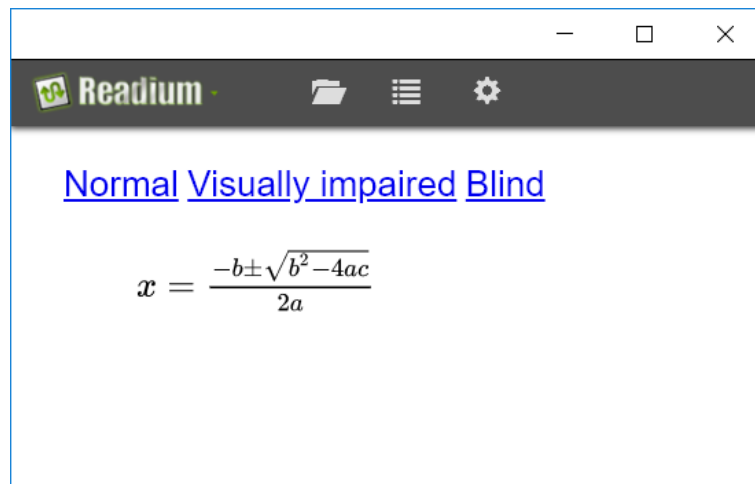
<figure>
  
  <p class="transparent">
    &lt;Image&gt;Indian rhino standing in tall grass in
    Kaziranga National Park, India&lt;/Image&gt;
  </p>
  <figcaption>
    Indian rhino in Kaziranga National park
  </figcaption>
</figure>

```

Figure 3.10: Code of figure 3.9 and 3.8

3.1.4 Mathematical formulas

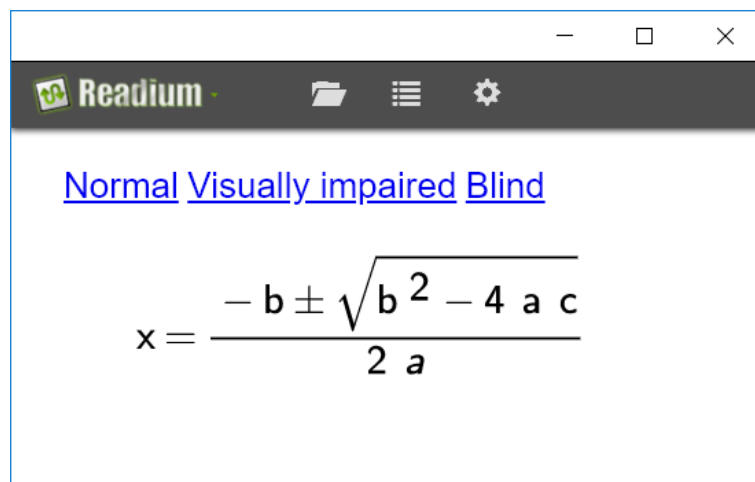
Formulas should be displayed in MathML. MathML is quite clunky and time consuming to write in, so a LaTeX to MathML converter has to be used. For the sample documents, an online converting tool was used. The input formula was the quadratic equation, which appeared in 1.10.



Normal Visually impaired Blind

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Figure 3.11: Equation in 'Normal' mode



Normal Visually impaired Blind

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Figure 3.12: Equation in 'Visual impairment' mode

In the normal version the default display style is used, which uses a serif font and a combination of italicized and unitalicized letters, as shown in figure 3.11. Serif fonts are not suitable in the visually impaired version, so a sans serif had to be chosen. At first the font was changed in the CSS. This unfortunately resulted in the text not scaling properly, and therefore some characters were not properly spaced. The root symbol also wasn't displayed clearly. Instead the mstyle attribute of MathML had to be changed to sans serif which resulted in figure 3.12. The mstyle attribute can not be changed in CSS, it has to be inserted into every math element.

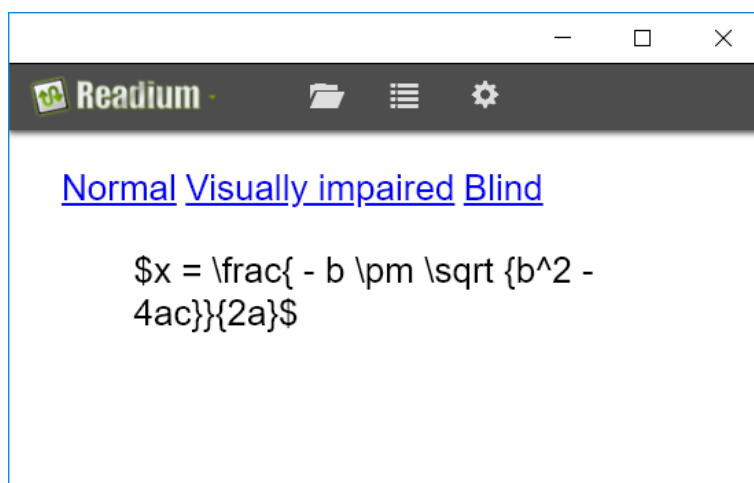


Figure 3.13: Equation in 'Blind' mode

In the blind version, the text has to appear as LaTeX code. Much like the alternative text for images, it will appear as a separate paragraph enclosed in `<p>` tags. To signify that it is code, it has to be surrounded by `$` signs, as shown in figure 3.13. The math element is in a `<div>` with CSS class "math", which is hidden in the blind version.

```

<figure>
  <div role="math" class="math">
    <math xmlns="http://www.w3.org/1998/Math/MathML" id="Formula" title
      ="Quadratic Formula" alttext="{x=\frac{-b\pm\sqrt{b^2-4ac}}{2a}}">
      <mstyle>
        <semantics>

          <mrow>
            <mi>x</mi>
            <mo>=</mo>
            <mfrac>
              <mrow>
                <mrow>
                  <mo>-</mo>
                  <mi>b</mi>
                </mrow>
                <mo>\pm</mo>
                <msqrt>
                  <mrow>
                    <msup>
                      <mi>b</mi>
                      <mn>2</mn>
                    </msup>
                    <mo>-</mo>
                    <mrow>
                      <mn>4</mn>
                      <mi>a</mi>
                    </mrow>
                  </mrow>
                </msqrt>
              </mrow>
              <mrow>
                <mn>2</mn>
                <mo></mo>
                <mi>a</mi>
              </mrow>
            </mfrac>
          </mrow>

        </semantics>
      </mstyle>
    </math>
  </div>

  <p class="transparent">
    $x = \frac{ - b \pm \sqrt {b^2 - 4ac}}{2a}$
  </p>
</figure>

```

Figure 3.14: Code of figures 3.11 and 3.13

```

<mstyle scriptsize="1" lspace="20%" rspace="20%"
  mathvariant="sans-serif">

```

Figure 3.15: Addition to the code shown in 3.14 for the visually impaired version to make the formula appear as sans serif and not in italics

4. Evaluation EPUB Standard

Normal Visually impaired Blind

Test File

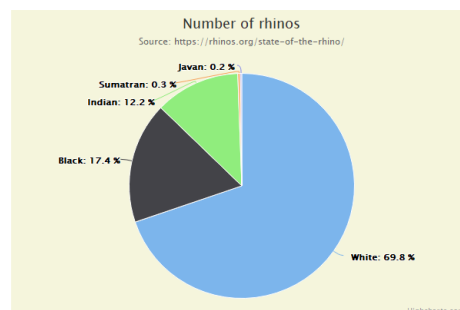
The quadratic equation is shown below.

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

- Bullet point 1
- Bullet point 2



The SZS logo



Percentages of each rhino species from the total rhino population

Figure 4.1: Example EPUB which will be tested

This chapter is going to present an evaluation of how documents of the new standard were displayed in several EPUB reading systems. The test document has the same content that will be in both versions of the new standard. The CSS version is shown in figure 4.1. It is shown in dual column mode so that the whole document can be shown at once. Actually the SZS logo is followed by the graph, so that is the proper reading order. The file has following content:

1. A heading of level 1
2. Standard text

3. A figure with a MathML equation
4. A short list
5. A figure with a PNG
6. A figure with a SVG

All of these features are important to keep the document accessible. It is vital to have both a PNG and a SVG to test if a SVG can be displayed. Unlike PNG, they are not a standard format so they might not be supported. The test device is a Nokia 8 with Android 8.1.

4.1 Reading systems on Android

4.1.1 Tolino

Tolino is a cooperation of several large German booksellers to create a unified E-Book reading device¹. There is a Tolino app on Android which can be used to see how well the EPUBs following the document standard run on the Tolino reading device family, as they likely share similar technology. However, the Android app is more powerful than the Tolino reading devices as it can also show color beyond greyscale. As seen in figure 4.2, the app is unable to display MathML. Furthermore, it does not show the PNG file, which should actually be supported, as other files with images are displayed properly. This is due to the image being enclosed in a figure element. If it is not in one, the image is displayed correctly.

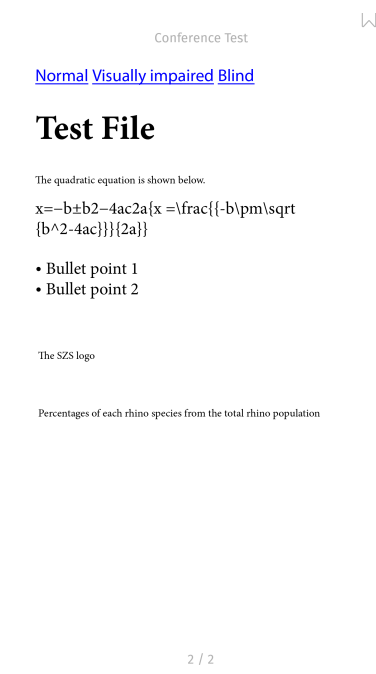


Figure 4.2: EPUB of the CSS standard shown in Tolino

The switching mechanism does not work with both the CSS and JavaScript versions. This means that the Tolino app does not support scripting and also does not support

¹<https://mytolino.de/tolino-kooperationen-und-vertrieb/>

CSS 3. The preinstalled screen reader of Android, TalkBack, also does not work with the Tolino app.

4.1.2 Reasily

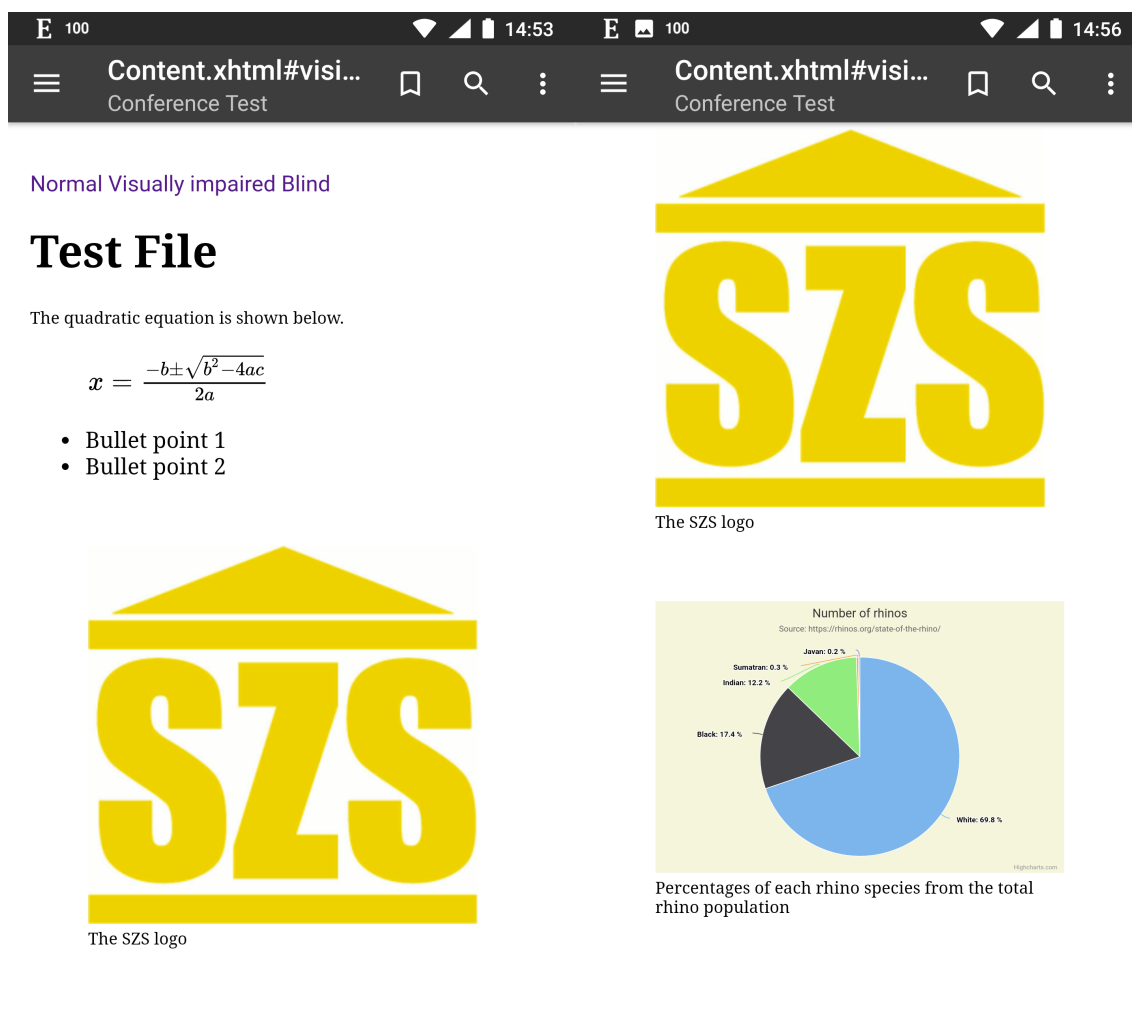


Figure 4.3: The normal display style in Reasily

Reasily is an Android EPUB reading app which states that it has support for several EPUB 3 features such as MathML and media overlay for read aloud books². While the JavaScript standard does not switch versions correctly, the CSS version does. The normal display style is shown in figure 4.3. All elements are shown as desired. Each of the figures with MathML, PNG and SVG are displayed properly.

The visually impaired display style is also shown properly, with the MathML equation using a sans serif font. The blind display style in figure 4.4 is shown correctly, with the equations and images showing the alternative text with the appropriate tags. Furthermore, Reasily supports TalkBack and elements such as the heading of level 1 are identified properly.

²<https://play.google.com/store/apps/details?id=com.gmail.jxlab.app.reasily>

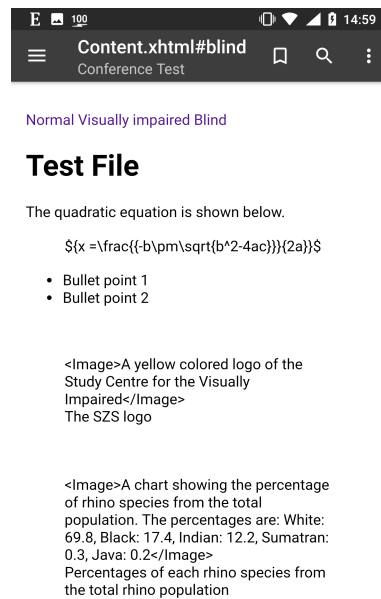


Figure 4.4: The blind display style in Reasily

4.1.3 Gitden Reader

Gitden Reader is an Android app³ with MathML support. It does not support both of the switching mechanisms, but does display the MathML, PNG and SVG properly. It supports TalkBack, but the equation is not identified properly and not read aloud.

4.1.4 FBReader

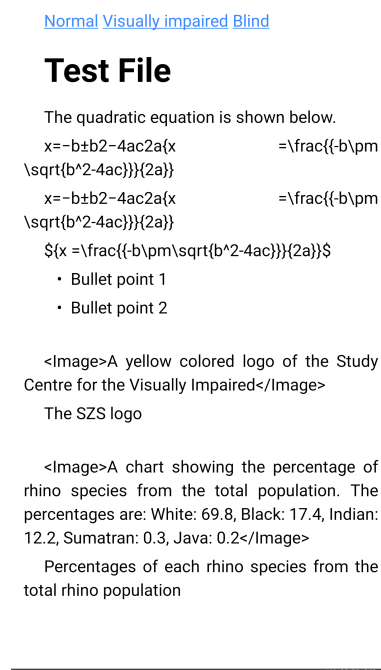


Figure 4.5: The CSS file in FBReader

³<https://play.google.com/store/apps/details?id=com.gitden.epub.reader.app>

FBReader is a popular E-Book reader available on Android⁴. Both CSS and JavaScript switching do not function as intended. Furthermore, none of the MathML, PNG and SVG are shown properly. The MathML first shows the equation not rendered properly and then annotation which should actually be hidden. One interesting observation is that elements which are supposed to be hidden, such as the text between the `<Image>` tags, are shown. The CSS file even has three pages showing the same page as shown in figure 4.5, only the links at the top are missing. Thus it is likely that FBReader does not support certain CSS features such as hiding elements.

4.1.5 UB Reader

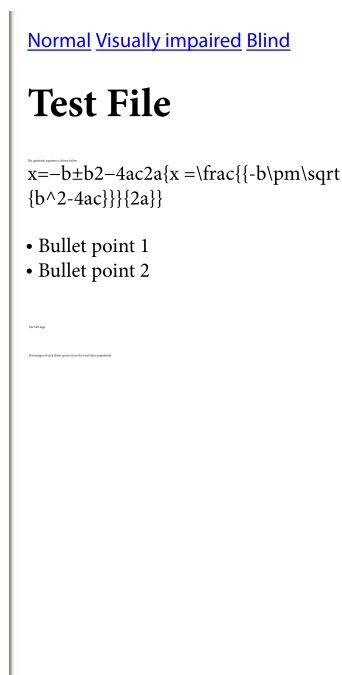


Figure 4.6: The CSS file in UB Reader

UB Reader (Universal Book Reader) is another popular Android E-Book reader⁵. The switching mechanism does not work with either standard and as shown in figure 4.6, the equation and the images are not shown. Strangely enough, the standard text is shown very small, while links, bullet points and other text is shown much larger.

4.2 Reading systems on iOS

4.2.1 iBooks

iBooks is a E-Book reading system create by Apple for their devices⁶. It does not support either one of the switching mechanisms, but can display MathML. Images are not displayed properly, and this likely due to them being in figure elements. The Apple screen reader, VoiceOver, can be used to read EPUB files.

⁴<https://play.google.com/store/apps/details?id=org.geometerplus.zlibrary.ui.android>

⁵https://play.google.com/store/apps/details?id=com.mobisystems.ubreader_west

⁶<https://www.apple.com/ibooks/>

4.3 Reading systems on Windows

4.3.1 Calibre

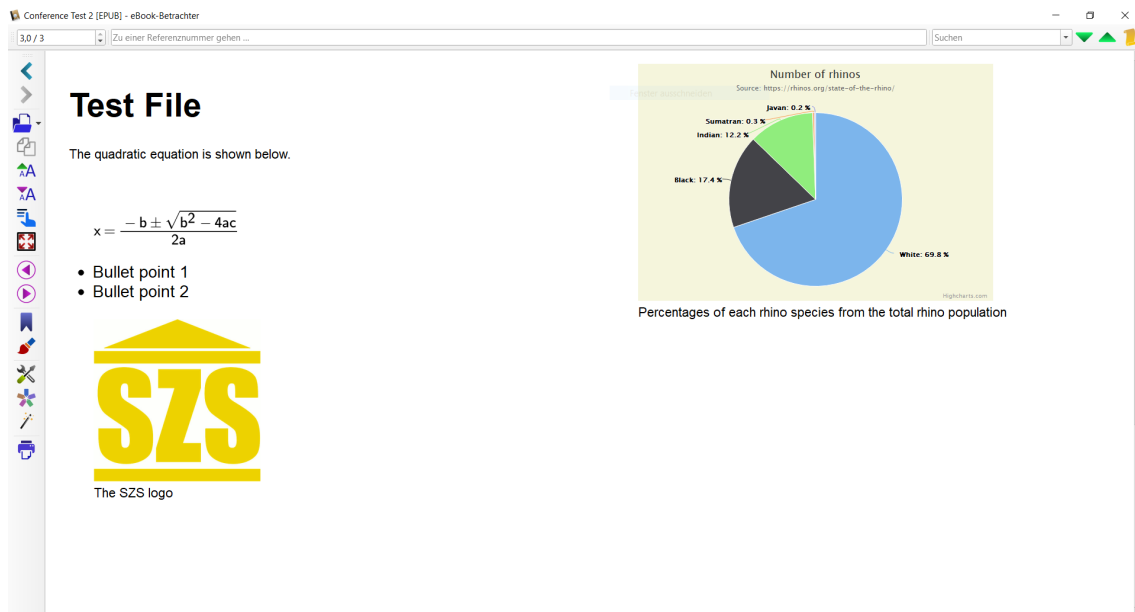


Figure 4.7: The visually impaired display style in Calibre

Calibre is an E-Book management system which is open source⁷. Calibre is the only reading system to support the JavaScript switching. It has session and not local storage, meaning that the display style is remembered until the EPUB is closed. When it is opened again, it does back to the default display style. As seen in figure 4.7 and 4.8, all elements are shown properly, and when the images are hidden in the blind display style, the alternative text is shown properly. The CSS display style is not supported. One major problem with Calibre is that screen readers are not supported, because it is impossible to access figures and text.

⁷<https://calibre-ebook.com/>

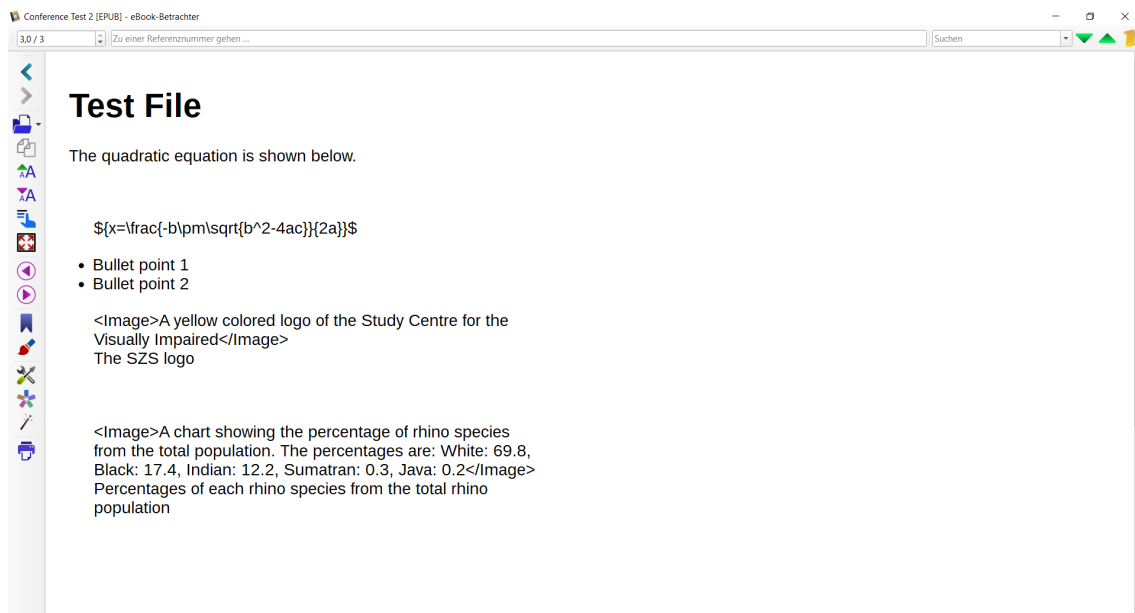


Figure 4.8: The blind display style in Calibre

4.3.2 Radium

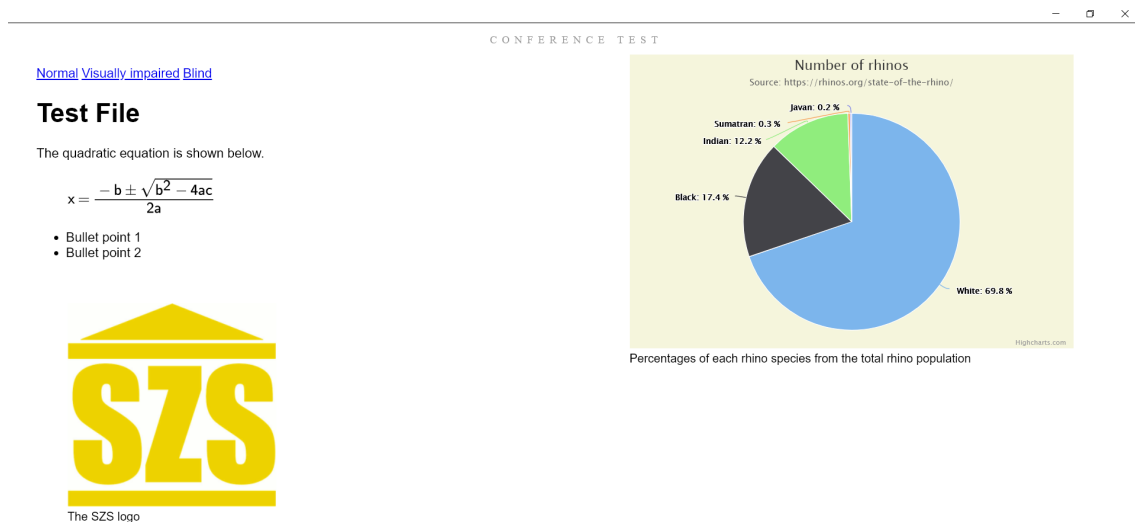


Figure 4.9: The visually impaired display style in Radium

Radium began as a project of the IDPF as an open source application to create reading systems which follow the newest EPUB standards⁸. As such, it supports many of the new features. As shown in figures 4.9 and 4.10, both the visually impaired and blind display style of the CSS standard are shown as intended. The JavaScript standard does not work. Furthermore, it supports screen readers and everything is identified correctly. However, there still is one issue. While testing several CSS EPUB documents, the switching in

⁸<http://readium.org/about/faq>

the short ones did not function properly. In the preview browser in Accessible EPUB and when the individual XHTML files were opened in web browsers the behaved correctly.

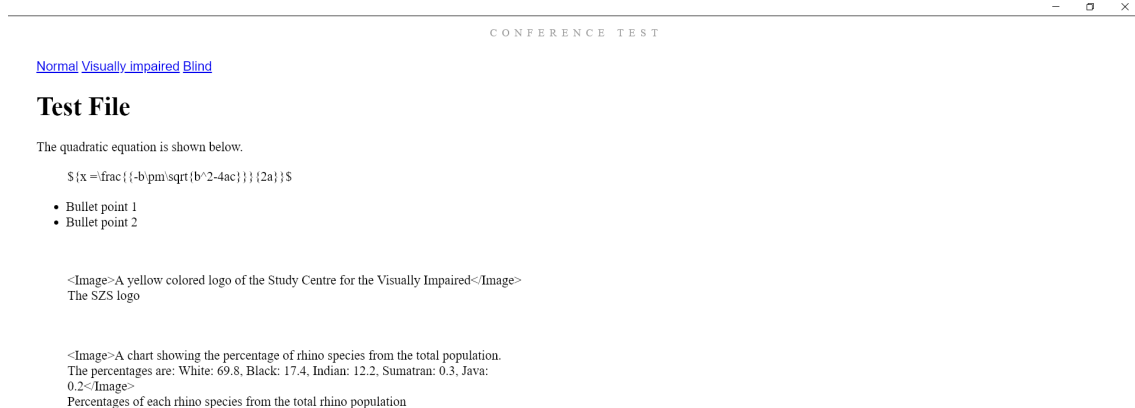


Figure 4.10: The blind display style in Radium

4.3.3 Bluefire Reader

Figure 4.11 of the Bluefire Reader⁹ is very similar to figure 4.2 of the Tolino app. They don't show the images and cannot display MathML properly. Both CSS and JavaScript switching do not work. Screen readers can also not read the content of the document.

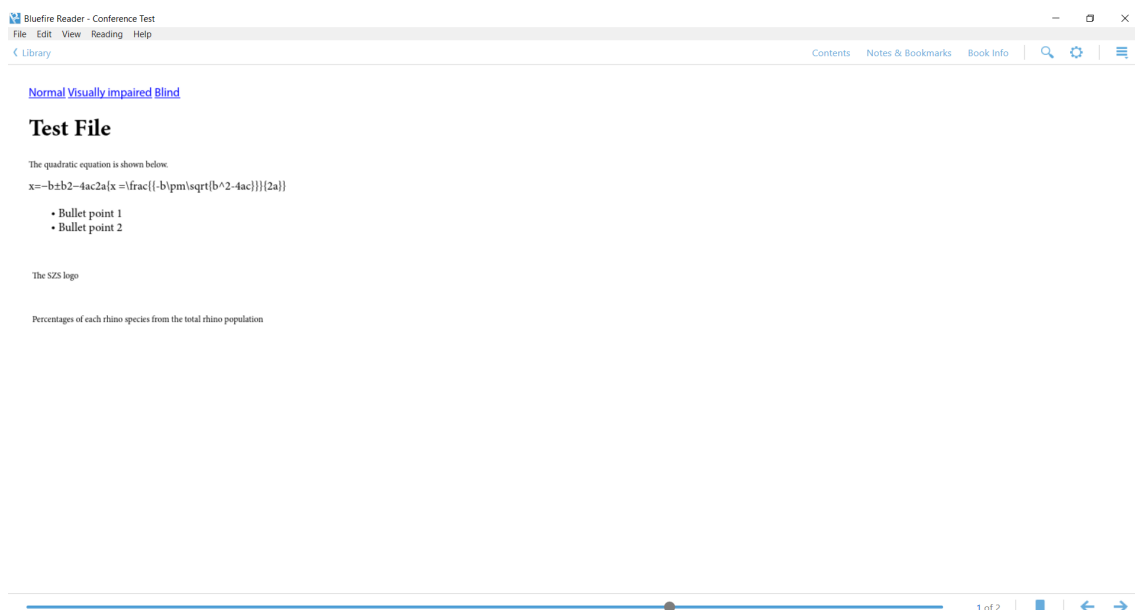


Figure 4.11: The CSS file in Bluefire Reader

⁹<http://www.bluefirereader.com/bluefire-reader.html>

4.3.4 Icecream Ebook Reader

Figure 4.12 shows the CSS test file open in Icecream Ebook Reader¹⁰. While MathML is not displayed properly the SVG and PNG are shown correctly. The switching mechanism does not work either with CSS or JavaScript and there is no screen reader support.

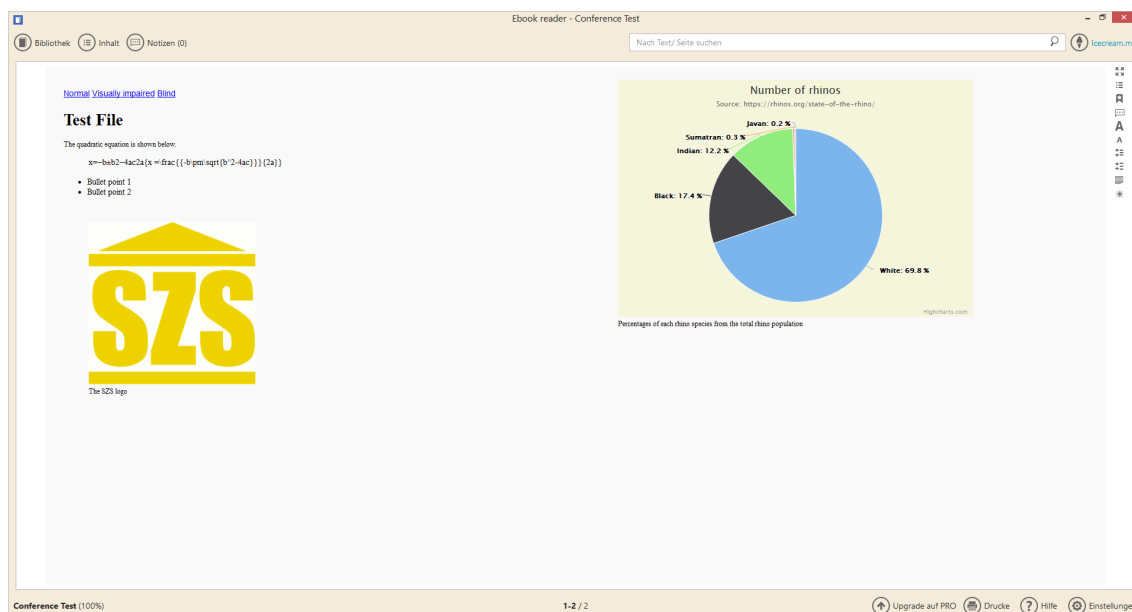


Figure 4.12: The CSS file in Icecream Ebook Reader

4.3.5 AZARDI

AZARDI Desktop¹¹ is a E-Book reader recommended by the DAISY consortium [6] as supports accessibility and it works with the JAWS screen reader. Furthermore, as seen in figure 4.13, it displays most of the content correctly, including MathML. The elements it does not display correctly are the links. They are shown as normal text and it is not indicated that they can be clicked. The switching mechanism does not work in both CSS and JavaScript.

¹⁰<https://icecreamapps.com/Ebook-Reader/>

¹¹<http://azardi.infogridpacific.com/>

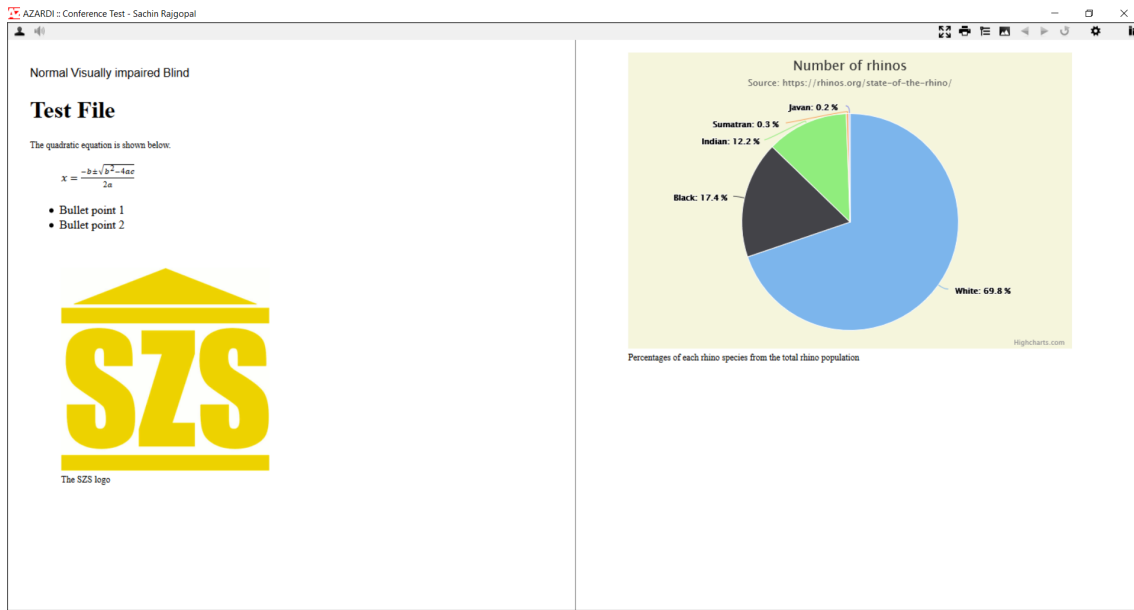


Figure 4.13: The CSS file in AZARDI

4.4 Discussion: what do developers have to do?

The EPUB 3 specification [12] was recommended on the 11th of October 2011. More than 6 years later, most of the EPUB reading systems do not support many of the new features, such as MathML and CSS 3. Developers have to start supporting the new features as soon as possible. The CSS standard only works in Reasily and Readium, while the JavaScript standard only works in Calibre. JavaScript is optional so developers don't have to necessarily support that standard, but CSS is part of the EPUB specification. It is important that the new CSS features will be supported soon, so that the CSS standard functions properly. Even Readium which was originally created by the IDPF does not support the CSS perfectly. It seems like EPUB 3 will remain the standard for a longer period of time as EPUB 3.1 specification was just recommended on the 5th of January 2017 and it was just a small update. So developers of EPUB reading systems can be assured that the EPUB 3 specification won't be completely overworked in the foreseeable future and having some security for the future.

5. AccessibleEPUB Editor

In this section a short overview of the Accessible EPUB editor will be presented with the requirements it should have. After that there will be an evaluation where the various problems and issues faced during the development will be explained and how they were solved.

5.1 Requirements

There are a variety of requirements which have to be fulfilled by a suitable editor. Most importantly, it has to be easily usable and not require any programming knowledge to use, except for LaTeX code for equations. Currently no other EPUB editor allows users to insert equations without them explicitly programming it in. The technology required for EPUB documents must be done behind the scenes and the user must not be exposed to it.

5.2 Programming language

The first question was in which programming language should the editor be implemented in. C# and Java were picked early as the two main options, as both of them are object oriented and natively support forms. Furthermore, both support the ability to make the programs themselves accessible, e.g. for blind users. C# has several accessibility properties, like AccessibilityDescription and AccessibilityName, which are passed to the screen reader. Java uses the Java Accessibility Bridge which makes it accessible to screen readers. Java is platform independent, and while C# programs can run on Mac OS and Linux, it relies heavily on Windows and its features. However, Java is not already installed on any operating system, while C# programs can run on Windows machine with only .NET as prerequisite. The target .NET version of the editor is contained in Windows 10. Therefore the editor was programmed in C#, as the users were predominantly Windows users and don't have to install any prerequisites.

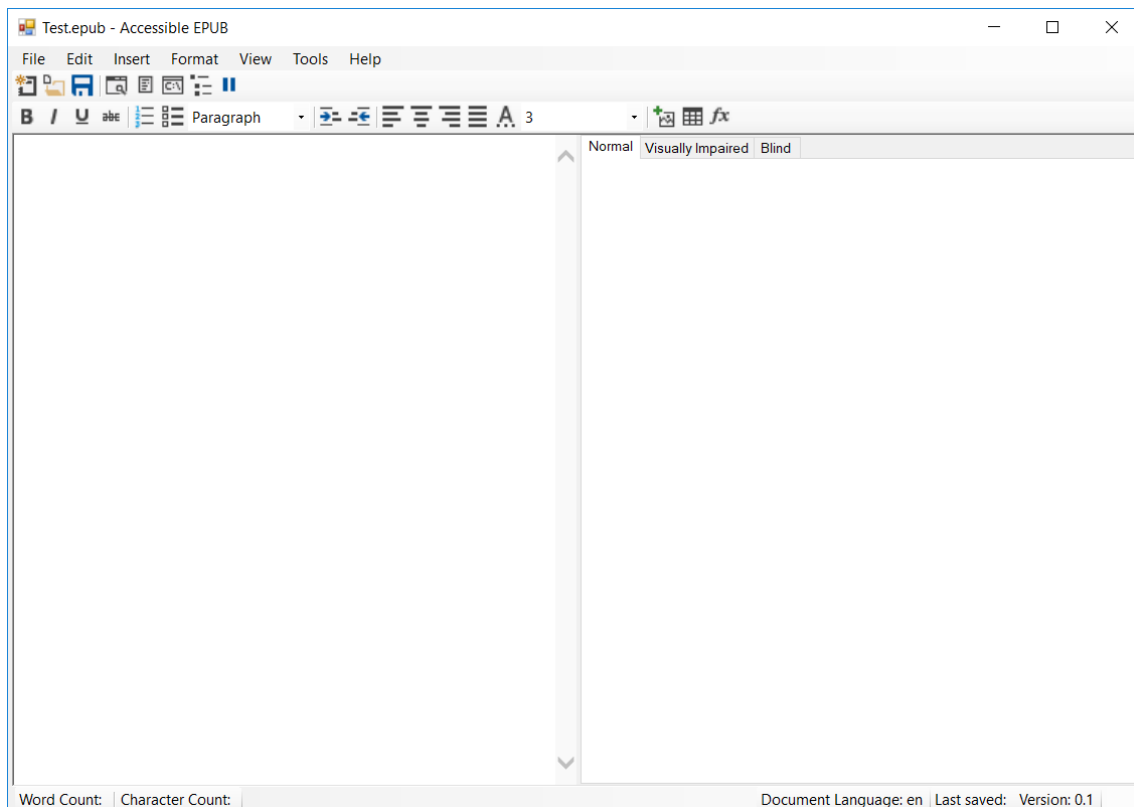


Figure 5.1: Main window with editor and JavaScript switching preview

5.3 Features

5.3.1 Main window with editor and preview

The main window is the entry point of the program, shown in both figures 5.1 and 5.2. At first it has a splash screen in main panel which then shows an editor and a web browser. The content typed into the editor is converted into HTML code which is then displayed in the browser and updated automatically. The web browser can display the content in each of the three versions mentioned in chapter 3. The toolbars at the top contain standard editor tools, such as making the text bold. It also contains some specialized tools, specific to this program. Some of them allow the user to add HTML elements like mathematics, tables or images. Other specialized tools allow the user to show and hide certain panels in the main form, such as the preview on the right hand side.

5.3.1.1 Editor

The main window required the most programming and therefore also had its fair share of problems. The first issue was regarding the editor on the left hand side in figure 5.1. At first a What-You-See-Is-What-You-Get(WYSIWYG) editor was not intended, as it hides semantic information about the EPUB. The initial approach involved editing semantic information of XHTML elements, like images, and changing things like the src, alt, etc. This was still in the early, rough stages and the approach seemed too complicated, so the WYSIWYG editor was chosen instead.

The second issue was how to implement the WYSIWYG editor. Programming it from scratch would not be possible in the time allocated to a bachelor thesis, as it would

involve creating a browser engine. Fortunately, the inbuilt browser in C#, WebBrowser, allows editing with just a few lines of code. The WebBrowser control is based on Internet Explorer and displays web pages like it. It uses Internet Explorer version 6 as default, which does not display certain CSS properties such as `max-width` and cannot display `s`. To fix this issue, some code has to be run when the form has loaded which determines the Internet Explorer version on the computer and loads the newest one to the editor. After this SVGs and the CSS properties functioned as desired.

A major issue with the editor is that content written in it is converted to HTML, while EPUB requires XHTML. While there aren't major differences, there are some small ones such as independent tags like `
` have to be self closing and written as `
` in XHTML. If this is not done the EPUB reader will specify an error. Therefore a tool has to convert the HTML code to XHTML. There are several packages available, one of them being `HtmlAgilityPack`¹. It can convert HTML and XHTML and also correct HTML parsing errors such as not closed tags. `HtmlAgilityPack` functioned well and as expected at first. The resulting code was in XHTML. However, there soon was an error. Many Greek signs, such as the sign "Λ", were displayed as question mark. This meant `HtmlAgilityPack` could not be used.

Another available package was `TidyManaged`², was also used, but it did not convert the code properly to XHTML. The third attempt involved using `SgmlReader`³, which converts SGML content, like HTML, to XML content, like XHTML. It successfully parsed the HTML and was able to convert mathematical signs properly. It did not create a XML declaration at the top of the document, but this was simpler to solve. A standard XML declaration was just added to the document and the new XHTML document was properly rendered by the browser.

5.3.1.2 Preview browser

On the right hand side of the main window is the preview browser. It should display the XHTML page in each of three versions(normal, visually impaired, blind). Since the HTML editor uses the WebBrowser control, it would have been easiest to use it on the right side too. However, Internet Explorer is unable to display MathML. Instead of showing the quadratic equation, as shown in figure 1.10, it showed figure 5.3. As a result, another preview solution had to be found. Only two browsers are able properly show MathML, Mozilla Firefox and Safari by Apple. The Firefox engine, Gecko, was chosen as it is open source and had a C# browser package. After inserting an instance of a `GeckoWebBrowser` in the window MathML was displayed properly.

As seen in figures 5.1 and 5.2, there is a small difference between the user interface of the JavaScript and CSS versions. The CSS version only uses one browser tab as preview, while the JavaScript version uses three. This is mentioned in chapter 3 and is due to the JavaScript version having a separate file named `VersionChanger.xhtml`, which handles switching versions. In the CSS version the whole content is in one XHTML file. Consequently, the CSS version is very easy to show in the preview. Only the three links shown in figure 3.3 have to be added to HTML editor content and file is done.

This is much more complicated in the JavaScript version, since the default display style will always be shown. The CSS of the blind and visually impaired version had to be

¹<http://html-agility-pack.net/>

²<https://github.com/markbeaton/TidyManaged>

³<https://github.com/lovettechris/SgmlReader>

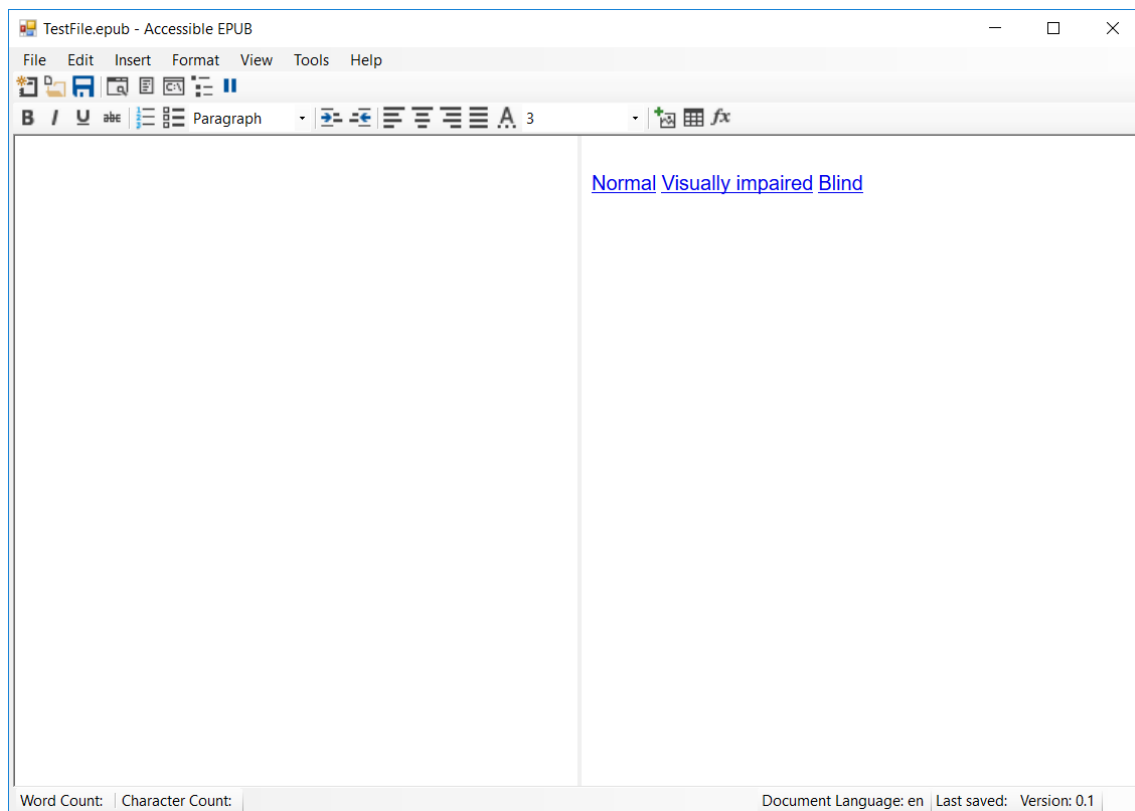


Figure 5.2: Main window with HTML editor and CSS switching preview

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Figure 5.3: MathML depiction of the quadratic equation in Internet Explorer

changed. The initial attempt to solve this problem was done by changing the CSS of each browser in a tab and accessing the `GeckoStyleSheet` of a browser. Unfortunately, even after changing the `GeckoStyleSheet` of a document, it still displayed the default one. An alternative approach consisted of accessing the head(<head>) of the XHTML document and inserting the CSS there, but this too did not work. These two methods were preferable as they do not have significant input and output (IO) operations. Unfortunately, even slightly altered methods of the two did not function properly. As such an IO intensive approach had to be taken. The XHTML file with the content, `Content.XHTML` had to be copied to two new files, `BliContent.XHTML` and `ImpContent.XHTML` for the blind and visually impaired version, respectively. Then the single line linking the style in XHTML head was replaced with one linking the corresponding CSS. Both files are then removed before saving the EPUB and then created again. Thus there is additional overhead, but it was the only method which succeeded.

An important feature of Accessible EPUB is updating the preview after a change in the HTML editor. Originally this was done only after saving the document to reduce overhead. However, this does fulfill the requirement of live updating the preview. A timer was created in the program which updates it every second. This produced substantial overhead, slowing the program down when large images are added. Changing the timer to five seconds addressed this issue.

After the preview is refreshed, it always get scrolled to the top. While this is not an issue in short documents, in long documents it is irritating to the user. The first approach involved saving the X axis and Y axis scroll bars' position, but it wasn't possible to get their values properly. A pseudo fix was designed instead which pauses the refresh after the user presses the pause button in the toolbar. This does not solve the issue, but it makes it less irritating to the user.

A feature which was also added was the ability to change the source code of the XHTML file. The user would be able to toggle between the HTML editor and the source code and edit both and the changes would then be reflected on both. While this seemed to work at first, additional testing exposed a major issue. When several tabs are open the content of the last tab is always overwritten from one of the other tabs. This issue was thought to be caused by the work which is done when the code of the HTML editor is converted to the code of the XHTML file. However, this didn't seem to resolve it. The issue only stopped appearing when the changes from source code editor were not saved and therefore lost. This issue wasn't solved as it is not a mandatory feature; the users of this editor likely do not have programming experience and changing the XHTML document might cause rendering problems when the EPUB is read.

5.3.2 Inserting mathematical equations

A focus of Accessible EPUB lies in the creation of documents for STEM subjects and must therefore allow users to insert mathematical equations easily. The equation editor is shown in figure 5.4 and the user must enter LaTeX code, as it is simpler to write than MathML and several conversion programs exist. The input is then shown in equation form in the lower panel, without the user entering any additional command. This is done with the .NET library package, `WpfMath`⁴. It parses the LaTeX in real time and gives immediate feedback to the user. In its default implementation, an incorrect input results

⁴<https://github.com/ForNeVeR/wpf-math>

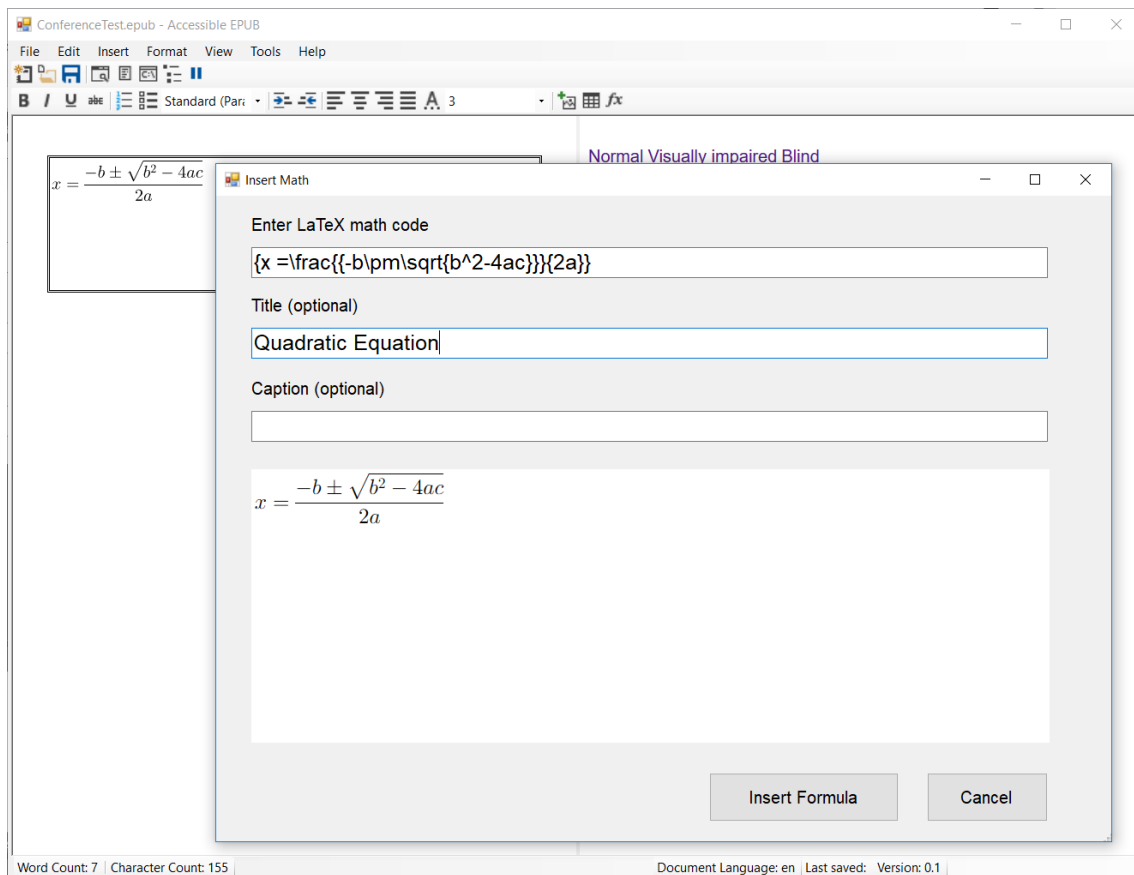


Figure 5.4: Accessible EPUB equation editor

in a blank panel. This might be confusing to the user, as they might not know where exactly the error in the input lies. Fortunately, WpfMath contains the method `HasError`, which detects if the input contains an error. This value was then passed to Accessible EPUB which then adds a red border around the panel, while leaving the parsed output of the last working input intact. The user is then able to get feedback to fix the issue while still having an idea where the problem was.

However, WpfMath does not convert the LaTeX code to MathML. No .NET library was found which converts LaTeX to MathML in a satisfying manner. Consequently, external solutions had to be found which follow the sequence of events.

1. LaTeX code is passed on to an external program
2. The external program converts the code to MathML
3. MathML is passed back to Accessible EPUB which then inserts it into the EPUB.

There were several factors which had to be considered when picking the external program. First of all, it had to be freely available, preferably open source, so that it can be distributed with Accessible EPUB. It should also require as few dependencies as possible. Lastly, it should not be a large program which would then greatly enlarge the file size.

Some of the programs considered were LaTeXML⁵, latex2mathml⁶ and TeX4ht⁷ among others. Unfortunately, all of have prerequisites, Perl, python and Tex, respectively. Finally, a program was found which satisfied the demands, pandoc⁸. It is not a very small program and with a size of 93.2 MB it does increase the size of the Accessible EPUB installation, but as pandoc can be run without dependencies this was deemed as an acceptable trade off.



Figure 5.5: Signs shown instead of \pm in pandoc

However, there were still some issues with pandoc. The plus-minus sign (\pm) does not get displayed properly when pandoc is run directly from Accessible EPUB. Instead of a plus-minus, the signs in figure 5.5 are shown. This also happens if a formula with a plus-minus sign is given as input parameter and pandoc prints the results of the conversion in the command shell of windows. This is an issue with the encoding the command shell uses. It cannot display some signs, such as even a simple minus sign (-). The conversion with pandoc also creates issues. The MathML specification [21] encodes several signs, including plus-minus. The encoded string is "±", which can be displayed the cmd shell. Pandoc does not use these encodings and uses the normal signs, which creates problems. The signs are shown normally is pandoc creates a stand alone HTML file with the `-s` option which can be viewed individually, but not when only the relevant MathML part is created. The problem is further compounded that using the output option, `-o`, of pandoc still produces the same encoding error. The signs are shown properly if the results are instead passed to a file with a redirect ("`>`") in the command shell. This means that pandoc cannot be called directly from Accessible EPUB and instead the command shell of Microsoft Windows, `cmd.exe`, has to be called as shown in figure 5.6. `accFile` refers to the file where the entered LaTeX code is saved to. It is then used here and the results, `formulaResult` is inserted in the inner HTML code of the editor. `CreateNoWindow` and `WindowStyle` are used to not show a command shell popping up. It used to be only shown for a moment, but it might confuse the user.

When the Accessible EPUB was installed on computers, there was another issue with pandoc. The file pandoc writes to was in the pandoc folder itself. While this worked before, it did not work when the editor was installed. The editor does not have permission to write to the pandoc folder, so it now writes to the folder for temporary files instead.

The MathML code from pandoc is then inserted in the inner HTML of the editor. However, Internet Explorer cannot display MathML and the editor display it as shown in figure 5.3. So a workaround had to be found. WpfMath can be used to create SVGs and PNGs of the mathematical formulas entered. So a SVG of the entered formula is generated and inserted into the editor with the MathML code. The CSS used by the editor hides the MathML and only shows the SVG. On the other hand, the preview browser hides the SVG and shows the MathML. The class `toRemove` of the image in figure 5.7 is not shown in the EPUB, but the division tag of HTML, `div`, with the class `math` is shown. The SVG is also used as an alternative image if the EPUB reading system

⁵<https://dlmf.nist.gov/LaTeXML/>

⁶<https://github.com/Code-ReaQtor/latex2mathml>

⁷<https://www.tug.org/tex4ht/>

⁸<http://pandoc.org/>

```

var proc = new Process
{
    StartInfo = new ProcessStartInfo
    {
        //FileName = "pandoc", //Path.Combine(pandoc, "pandoc"),
        FileName = @"c:\windows\system32\cmd.exe",
        Arguments = @"/c pandoc --mathml " + accFile + " > " +
            formulaResult,
        //UseShellExecute = false,
        //RedirectStandardOutput = false,
        CreateNoWindow = true,
        WindowStyle = ProcessWindowStyle.Hidden
    }
};

```

Figure 5.6: The C# code to execute pandoc

```

<FIGURE>
<IMG title="Quadratic Equation " class="toRemove" alt="{x=\frac{-b\pm\sqrt{b^2-4ac}}{2a}}" src="C:\Users\Sachin\AppData\Local\Temp\AccessibleEPUB\ConferenceTestJs\OEBPS\Images\Quadratic Equation.svg">

<DIV class="math" role="math">
<math title="Quadratic Equation" xmlns="http://www.w3.org/1998/Math/MathML"
altimg="C:\Users\Sachin\AppData\Local\Temp\AccessibleEPUB\ConferenceTestJs\OEBPS\Images\Quadratic Equation.svg"
alttext="{x=\frac{-b\pm\sqrt{b^2-4ac}}{2a}}">
...
<P class="transparent">${x=\frac{-b\pm\sqrt{b^2-4ac}}{2a}}$</P></FIGURE>

```

Figure 5.7: The inner HTML of the editor showing how a SVG is shown instead of the MathML

doesn't support MathML and supports the `altimg` attribute, as shown in the `<math>` tag in figure 5.7.

5.3.3 EPUB work behind the scenes

While the content documents of an EPUB are the most important part, EPUBs require the package file referenced in 1 to be updated. The user should not have to do anything with the package file, named `content.opf`. The first section `metadata` is filled in when a new document is created and the new file dialog box is shown, as seen in 5.8. Accessible EPUB requires the title and author, while the EPUB specification does not require an author. In many EPUB reading systems the data used to identify a book normally consists of both the title and author so it is mandatory here. The publisher is not required, but will be useful to identify material from educational institutions. The only language choices currently available are English and German, but more languages will be added later in the development.

The `manifest` declaration contains all of the files in the EPUB document. Files not declared might be shown properly. The user should not have anything to do with the manifest and it should be updated automatically. This is done when the user saves the file. The program looks through the whole folder and checks the file type and enter both it and the location into the manifest. An example is shown in figure 5.9. The `id` element is taken from the file name. Only the navigations documents have predefined ones. The `href` element is the path from the package document. The `media-type` is determined from the file type and the appropriate declaration found in the EPUB specification.

The dialog box is titled "New File". It has a standard Windows-style title bar with minimize, maximize, and close buttons. The main area contains the following elements:

- Title:** A text input field.
- Author:** A text input field.
- Publisher (optional):** A text input field.
- Language:** A dropdown menu currently showing "English".
- Choose File Format:** A section with two radio buttons:
 - ☒ Standard - EPUB (JavaScript) (with an information icon)
 - ☐ EPUB (CSS) (with an information icon)
- Buttons:** "OK" and "Cancel" buttons at the bottom right.

Figure 5.8: The dialog box shown when a new file is created

The `spine` section is already predetermined. If the support for multiple content files is added, then the ability to change the spine and by extension the reading order will be included in Accessible EPUB.

5.3.3.1 Inserting images tables

Accessible EPUB allows the user to insert tables with the dialog box shown in figure 5.10. The bottom half of the figure shows the `C#` element, `DataGridView`. The size of it changes whenever the focus leaves the row or column text field by pressing the tab key or just presses elsewhere with the mouse. The user can also press the "Adjust Table" button. The alternative would have been to change the size whenever the values in the text fields are changed, but a useful feature is that values entered in the `DataGridView` are not lost if the size is increased or reduced. This allows for more flexibility in the table creation.

The user is also able to add images with window shown in figure 5.11. The three mandatory elements are the location of the an image, a title and some alternative text. The title allows users with screen reader to identify the image and skip it if desired. Alternative text is also a must to maintain accessibility for all users. Captions are optional and are always inserted below the image. In future it is planned to allow users to add the caption above the image too. An important accessibility feature mentioned in [4] is the use of image tag types. The choice of image tags automatically adjust to the current document language, irrespective of the program language. There are a few default tag types, but the user can type their own tag types in.

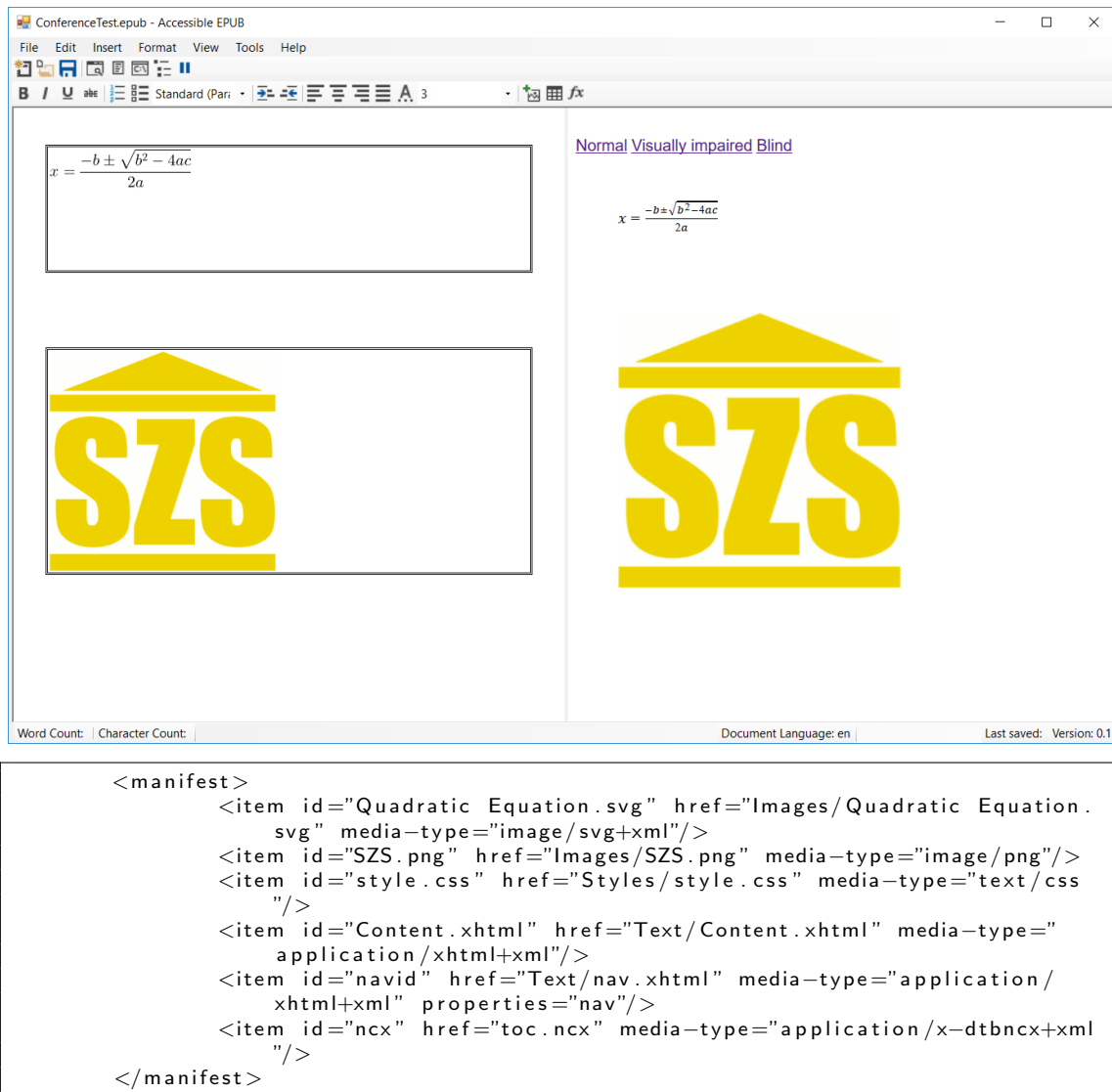


Figure 5.9: A sample EPUB of the CSS document standard and its corresponding manifest

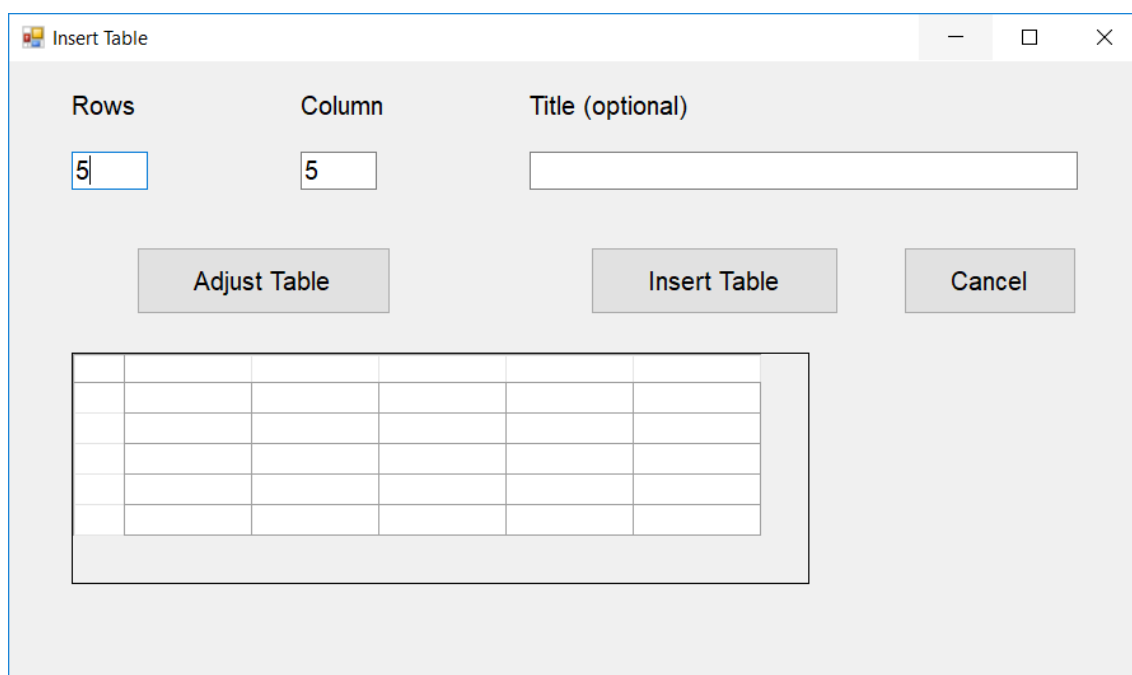


Figure 5.10: The dialog box shown when the user wants to insert a table

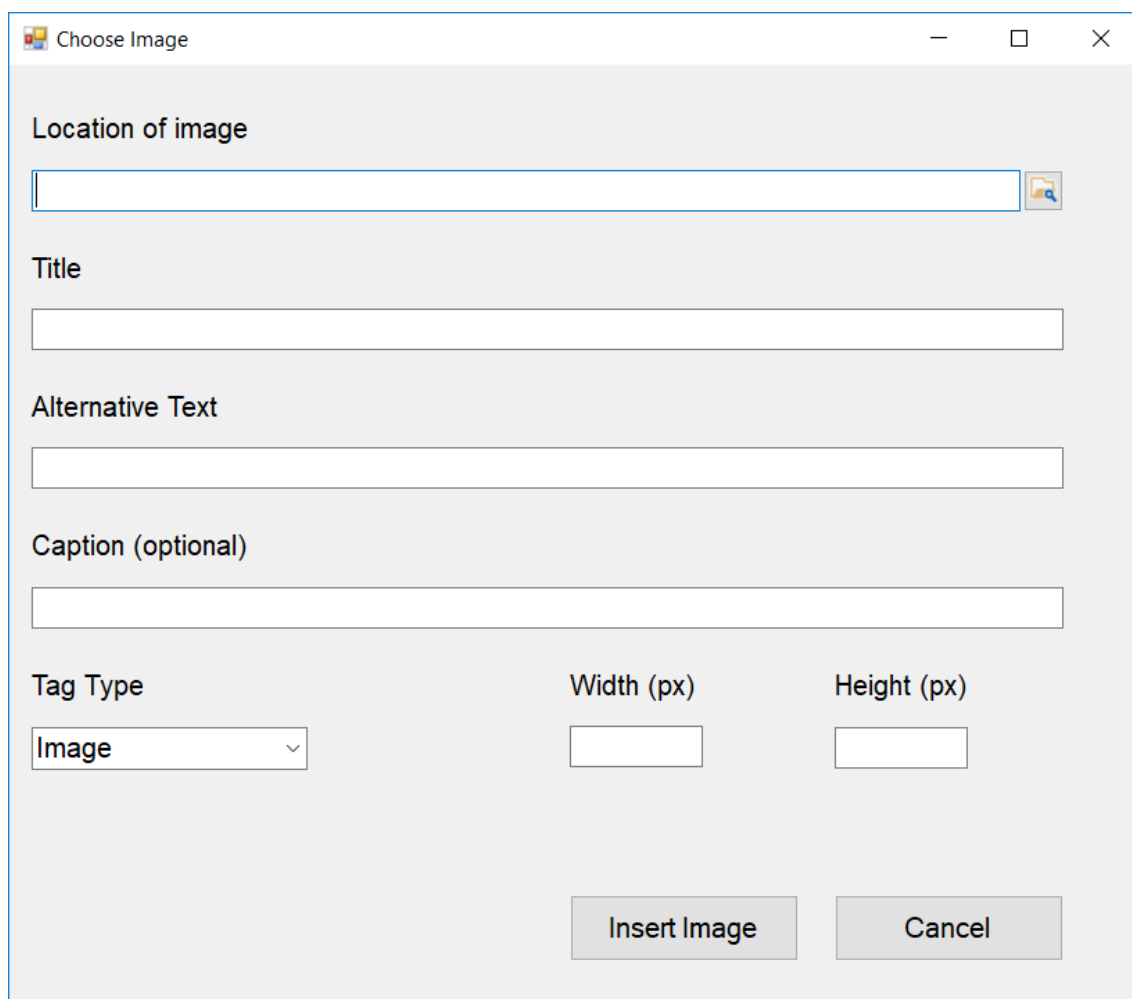


Figure 5.11: Insert image window of Accessible EPUB

6. Future Work

The Accessible EPUB editor is in a functional state where accessible EPUBs of the both new document standards, CSS and JavaScript can be created and edited. However, there is still much to be done.

The JavaScript document standard supports multiple HTML files as content documents. If the reading system supports local or session storage, it remembers which display style is to be displayed across several HTML files. This is actually the case with JavaScript standard in its current state as the first page with version switching mechanism is on a different page from the actual content. Accessible EPUB does not yet support multiple content pages. One issue mentioned in chapter 5 is there are issues with long documents. The preview browser automatically scrolls to the top as soon as the user writes something in the editor. If Accessible EPUB allows the user to add another content page, this problem can be avoided. Furthermore, the user can create documents with several sections and these can be managed individually. This does not work with the CSS standard as CSS changes are local to a HTML file and do not persist across multiple of them.

Future development will focus on usability and simplified import of other document standards. Allowing users to import files of different formats will allow an easier transition to Accessible EPUB. The features will be implemented for two formats at first: text documents and HTML files. Importing text documents is quite simple as the file just has to be read and the contents have to be inserting into the editor. Importing HTML files is a bit more complicated. Instead of inserting into the editor, the back end web browser of the editor must be inserted to. Images and equations will not be in the form of the accessible EPUB document standard, so they have to be converted into the correct form. The program will present the user with a process to examine every image and equation and insert alternative text when necessary. Other formats can be imported with the tool pandoc converting them. It is already a part of Accessible EPUB so it does not require any additional programs.

Another feature advanced users may want is a source code editor. As mentioned in chapter 5 a code viewer is already available, but it currently displays the entire file with the inner workings. It is also disabled as there were several issues with it. If a code

editor is included, the user should not be able to see the inner workings of the document standard, since editing those might break the EPUB file and make it unreadable.

To improve accessibility for visually impaired users, it should be possible to add an alternative image which can be a high contrast version of the original image. In addition to the one image document creators have to add, there will be an option to another one. This image will only be shown in the visually impaired display style. If no secondary image was chosen, the regular image will be displayed instead.

7. Conclusion

A short introduction is given where the motivation of this project and a short overview of the file format EPUB is presented. An EPUB document standard has been developed to allow visually impaired, blind and normal-sighted users to use the same document while meeting their respective accessibility requirements. There are two different versions of it, one using JavaScript and the other using CSS. Both versions of the document standard were tested on several reading systems. Only very few of the reading systems were able to show MathML equations and even fewer were able to display the document standards properly. EPUB 3 is not widely supported by reading systems.

An editor called "Accessible EPUB" was created so that users without programming knowledge can create accessible documents. It allows the user to write text much like in a word processor and insert mathematical equations, images and tables. The document standard uses EPUB 3, which is not yet fully supported by most EPUB reading systems. It is planned to add features to improve the usability of Accessible EPUB, such as importing files of different formats and further tools for accessibility.

Bibliography

- [1] PDF Association. *PDF/UA: The ISO standard for universal accessibility*. <https://www.pdfa.org/pdfua-the-iso-standard-for-universal-accessibility/>, visited on 23.03.2018.
- [2] Valentina Bartalesi and Barbara Leporini. "An Enriched ePub eBook for Screen Reader Users". In: *Universal Access in Human-Computer Interaction. Access to Today's Technologies*. Ed. by Margherita Antona and Constantine Stephanidis. Cham: Springer International Publishing, 2015, pp. 375–386. ISBN: 978-3-319-20678-3.
- [3] Deutsche Zentralbücherei für Blinde (German Central Library for the Blind). *Was ist DAISY? (What is DAISY?)* https://www.dzb.de/index.php?site_id=7.8, visited on 09.03.2018.
- [4] Verband für Blinden-und Sehbehindertenpädagogik e. V. (VBS) (Association for teaching for blind and visually impaired students). *Augenbit Wiki*. <http://www.augenbit.de/wiki/index.php?title=Hauptseite>, visited on 09.03.2018.
- [5] Antonello Calabrò, Elia Contini, and Barbara Leporini. "Book4All: A Tool to Make an e-Book More Accessible to Students with Vision/Visual-Impairments". In: *HCI and Usability for e-Inclusion*. Ed. by Andreas Holzinger and Klaus Miesenberger. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 236–248. ISBN: 978-3-642-10308-7.
- [6] DAISY Consortium. *AZARDI: EPUB reader for Windows*. <http://www.daisy.org/daisypedia/azardi-epub-reader-windows>, visited on 25.03.2018.
- [7] DAISY Consortium. *Baseline for Accessible EPUB3*. www.daisy.org/baseline, visited on 29.01.2018.
- [8] Matt Garrish and Markus Gylling. *EPUB 3 Best Practices*. O'Reilly, 2013. ISBN: 1449329144, 9781449329143.
- [9] John Gruber. *Markdown*. <https://daringfireball.net/projects/markdown/>, visited on 08.03.2018.
- [10] University of Hagen. *PDF- und Word Dokumente barrierefrei umsetzen (Making accessible PDFs and word documents)*. https://www.fernuni-hagen.de/barrierefrei/pdf_word.shtml, visited on 09.03.2018.
- [11] IDPF. *EPUB 3 Changes from EPUB 2.0.1*. <http://www.idpf.org/epub/30/spec/epub30-changes.html>, visited on 29.01.2018.
- [12] IDPF. *EPUB 3 Specification*. <http://www.idpf.org/epub/301/spec/epub-overview.html>, visited on 29.01.2018.

- [13] IDPF. *EPUB 3.1 Changes from EPUB 3.0.1*. <http://www.idpf.org/epub/31/spec/epub-changes.html>, visited on 09.03.2018.
- [14] IDPF. *Open Publication Structure (OPS) 2.0.1 v1.0.1 Recommended Specification*. http://www.idpf.org/epub/20/spec/OPS_2.0.1_draft.htm, visited on 09.03.2018.
- [15] IDPF. *Understanding EPUB 3*. <http://epubzone.org/epub-3-overview/understanding-epub-3>, visited on 29.01.2018.
- [16] Dublin Core Metadata Initiative. *About the Dublin Core Metadata Initiative*. <http://dublincore.org/about/>, visited on 23.03.2018.
- [17] John MacFarlane. *Pandoc document converter*. <http://pandoc.org/>, visited on 08.03.2018.
- [18] WebAIM (Web Accessibility in Mind). *World Laws: Introduction to Laws Throughout the World*. <https://webaim.org/articles/laws/world/>, visited on 30.01.2018.
- [19] Thorsten Schwarz. *Lecture notes in Accessibility - Assistive Technology for Visually Impaired Persons*. 2017.
- [20] Jens Voegler, Jens Bornschein, and Gerhard Weber. "Markdown – A Simple Syntax for Transcription of Accessible Study Materials". In: *Computers Helping People with Special Needs*. Ed. by Klaus Miesenberger et al. Cham: Springer International Publishing, 2014, pp. 545–548. ISBN: 978-3-319-08596-8.
- [21] W3C. *MathML Fundamentals*. <https://www.w3.org/TR/MathML2/chapter2.html>, visited on 23.03.2018.
- [22] W3C. *Web Content Accessibility Guidelines (WCAG) 2.0*. <https://www.w3.org/TR/WCAG20/>, visited on 29.01.2018.
- [23] W3Schools. *CSS Selectors Reference*. https://www.w3schools.com/cssref/css_selectors.asp, visited on 15.03.2018.