Flex Picture eBook Builder - Simplifying the Creation of Accessible eBooks

Danya Gharbieh[1], Maximilian Punz[1], Klaus Miesenberger[1], and Valentín Salinas-Lopez[1]

Institut Integriert Studieren, Altenbergerstr. 69, 4040 Linz, Austria
{danya.gharbieh, maximilian.punz, Klaus.Miesenberger, valentin.salinas_lopez}@jku.at

Abstract. The FPB project leverages the latest advances in EPUB3 to transform the creation of inclusive digital learning materials. While EPUB3's support for rich media holds immense potential for accessible education, widespread adoption is hindered by a steep learning curve and the time-intensive nature of development. This paper presents FPB's approach to streamlining the authoring of interactive, adaptable EPUB3 publications. FPB empowers the creation of illustrated digital books that dynamically tailor themselves to individual children's needs, fostering equal access to foundational education.

Keywords: Accessible digital publishing · EPUB3 · Accessible digital illustrations

1 Introduction

This paper introduces a novel approach to enhance the accessibility and usability of digital publication through adaptable non-text elements. The EU-funded Flex Picture eBook (FPB) project strives to revolutionize the adaptability of digital publications, particularly illustrated children's books. By testing a prototype publication with children with disabilities, their care providers, and educators, the FPB project aim to improve inclusion in digital publishing.

We present a software suite designed streamlined authoring. It offers innovative tools addressing two core areas:

- Accessible Illustration Creation: Assisting authors in generating illustrations that are inherently designed to accommodate a range of comprehension levels.
- EPUB3 Integration: Providing mechanisms to seamlessly incorporate complex and layered illustrations into compliant EPUB3 publications, ensuring compatibility with common consumer electronic devices and assistive technologies.

By addressing these distinct yet interconnected challenges, the FPB project paves the way for mainstreaming accessibility in the publishing sector.

2 State of the Art

Many word processing tools such as *Google Docs*[3] now support exporting text files to the EPUB format. However, this only allows users to convert static files, and excludes the possibility of handling user input, thus making it unsuited for the creation of Flex Picture eBooks.

Currently, these kinds of eBooks have to be made by hand, which poses a lot of difficulties that could potentially deter authors and publishers from distributing their work in this new accessible format. The main problem is the technical nature of the EPUB file type[1]. Under the hood, they are specially formatted ZIP files, containing XHTML pages. All of the text has to be manually input into these XHTML files and styled through CSS, while the interactive functionalities have to be coded in JavaScript. In some way, each page can be thought of as an individual website that has to be manually programmed. This poses a serious technical challenge for the publishers, the majority of which do not possess a background in programming.

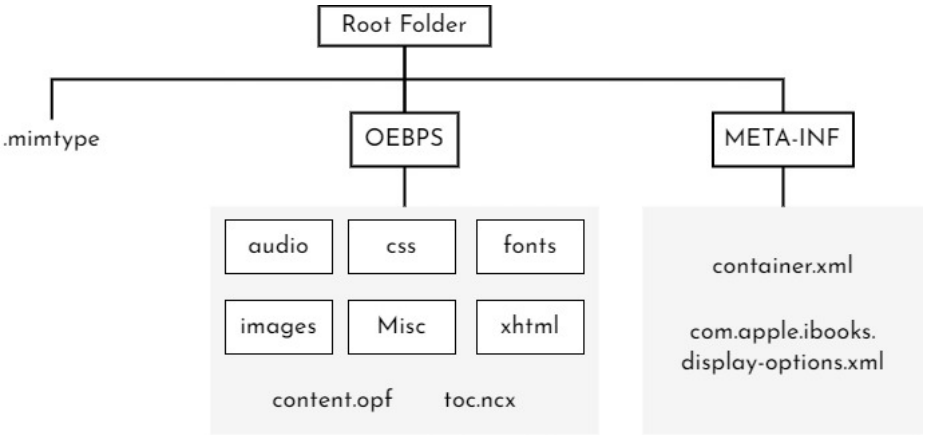

Fig. 1. The file structure of the EPUB Format. The mimetype tells the e-Reader that it is a digital publication, while META INF contains files which point towards the root documents. The OEBPS directory contains the folders in which all of the files inside the book are stored (sorted into the corresponding subfolders).

The EPUB format also has very strict guidelines that have to be fulfilled. If even a single condition is disregarded, or a parameter is altered in an unexpected way, the EPUB cannot be displayed on an otherwise compatible screen reader[8]. This brittleness spans multiple dimensions, such as the way the files are arranged inside the containers. There is a required folder structure that dictates what kind of files are put into which directory (see fig 1). All of these file structures have to be created, and even subtle errors such as misspelling a folders name can be detrimental. The format also requires a couple of different files that point the

e-Reader to the location of the pages are and help it navigate its contents. These files also have to be written in very particular ways, requiring a number of script tags and specifications to be working properly. IDEs and Word Processors, in which these files are usually created, do not tell the user which tags are still missing or what options are allowed, which makes working on these files very opaque and can often times lead to great frustration.

There is specialized software for testing EPUBs, such as the *EPUB Checker*[4], that allows the user to validate their files. Their purpose is to draw attention to the flaws present in the documents and inform them of specific problems. This makes the creation process slightly more transparent, giving the users a list of errors they can fix. But, to make the files compliant to the standard, using a validator still requires knowledge about technical details and can be very time consuming.

Software such as *Sigil*[6], helps navigate this maze somewhat, letting users generate the directory structures and required files. But, using them still is fairly demanding on a technical level, making it inaccessible for people who have no experience working with web technologies. Because, like with the examples of other text processors, the output of these files is static, and the additional Javascript code needed for the creation of Flex Picture eBooks has to be implemented manually.

Our proposed software should lower the barrier of entry that comes with the creation of Flex eBooks, by automatically allowing users to generate compliant EPUBs that include the Flex-eBook functionality, with the user having to interact with the coding as little as possible, while giving them access to the full range of functionalities that come with the format.

3 Methodology

In the section below, the methodology behind creating the Flex Picture eBook Builder will be explained. The creation-process was split into two separate, but interconnected parts: The user interface and the system that creates the EPUBS themselves.

3.1 Building an Accessible User Interface

Creating an accessible user interface (UI) is crucial for ensuring that all users, including those with disabilities, can interact with digital products effectively and comfortably. To make the software available to as many people as possible, we decided to use Electron[2] as the development framework, because it allows for the creation of platform independent applications. In the section below, detailed explanations of all of the screens that the user is presented with can be found.

Project This is the starting point of a FPB publishing journey. Here, the user has the opportunity to manage existing projects or begin new ones. (see fig 2)

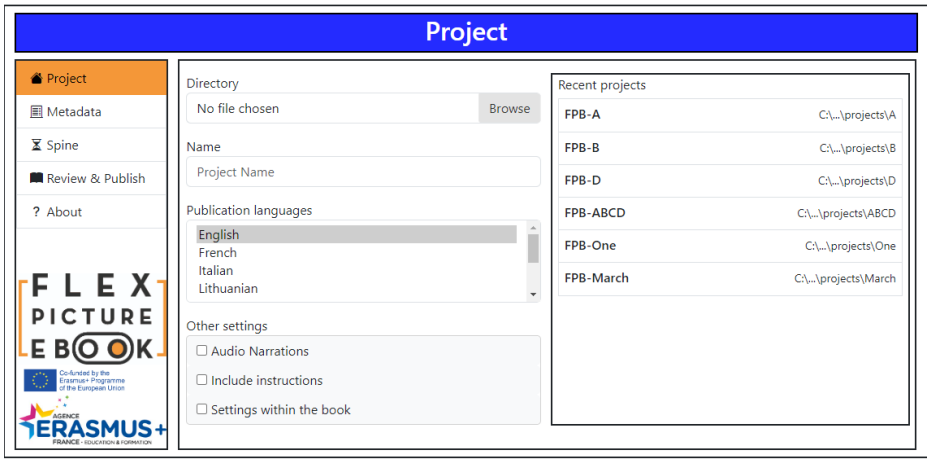

Fig. 2. Project Screen

- Manage Existing Projects: utilize the 'Recent Projects' section located on the right-hand side of the interface. This feature provides convenient access to previous projects.
- Start a New Project: utilize the main space for:
  1. Directory: browse to select the directory to store the project files.
  2. Name: provide a descriptive name for the project.
  3. Publication languages: Specify the languages in which the EPUBs will be created.
  4. Other settings: customize settings for the book, whether it is going to include options like audio narrations, instructional content, and if the settings will be included.

Metadata Here, the user provides the essential information that describe their FPB. (see fig 3)

- Add metadata: choose a type of metadata from 'Available Metadata' on the left, then click "Add" to see the required fields on 'Added Metadata' on the right side.
- Remove metadata: choose item from the 'Added Metadata' list on the right, then click "Remove" to return the fields back to available metadata list on the left side.
- Added Metadata language-specific types: some types of metadata that need multilingual input (e. g. the title) will have editable table spaces for each of the previously selected publishing languages.
- Required types: The application highlights the required types and prompt the end user to complete them before proceeding.
- Info section: shows information about the selected type of metadata and how to set it properly.

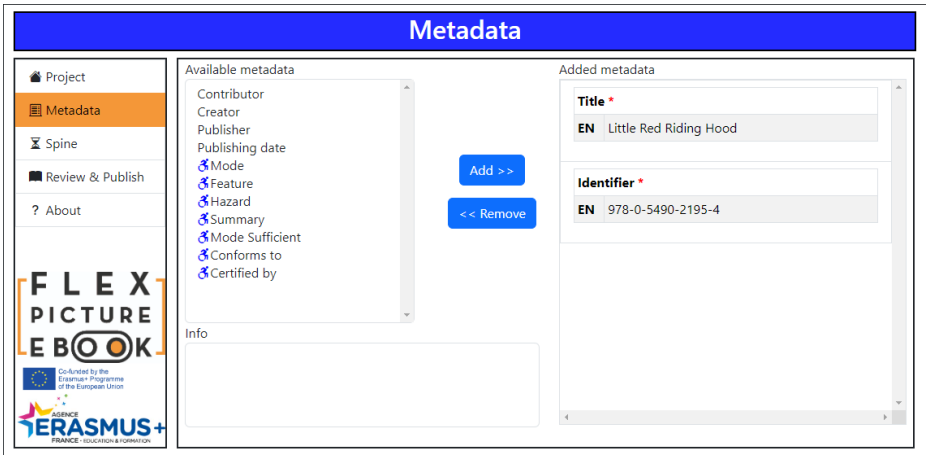

Fig. 3. Metadata Screen

Spine In this section, all content elements can be added and arrange the pages of the FPB. (see fig 4)

- Add and order pages within the publication, excluding the cover (first page) and credit (last page).
- Content section for each page:
  1. Text: enter the page text content in the editable text table for each of the publishing languages.
  2. Narrations: upload audio narration files for each of the publishing languages.
  3. Images and Scripts:
     (a) Image: Browse to select the XHTML file containing the adapted illustration (SVG).
     (b) Style: The application will identify external linked resources such as CSS and JavaScript files and prompt the user to specify their locations for proper integration and rendering within the project.
     (c) Verify linked files and avoid unnecessary duplication of resources used across multiple pages.
  4. Alternative text: enter the image alt text in the editable table for each of the publishing languages.
  5. Remove: Delete the entire selected page, excluding the cover (first page) and credit (last page).

Review and Publish In the final section, the FPBs are finalized and prepared for distribution. (see fig 5)

- File name: Set the publication file name, By default, it will contain the same value entered in the 'Name' field on the project screen.

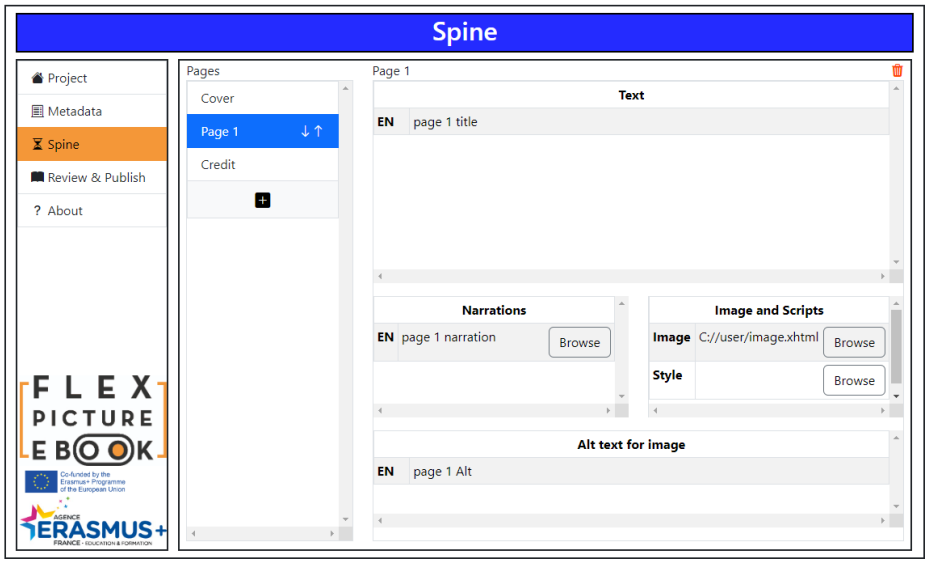

Fig. 4. Spine Screen

- Launch published E-Book: Tick the box to run the eBook immediately (using Thorium).
- Generate: Create the intermediate file structure and the metadata files. All generated content will be placed in a set of subfolders (one per publishing language) in the project root directory. This allows the user to examine and modify the files that will build up the FPBs if needed.
- Review: Validate the generated file structure against EPUB3 standards using EPUB checker tool.
- Publish: Outputs an EPUB from the generated files for each publishing language that is ready for distribution.

3.2 File Management and Automated Flex-eBook Creation

One of the more interesting challenges that we ran into was automatically importing files that are required for the interactivity to work. The images are saved as SVGs inside XHTML files, and often need additional Javascript code, or styling and formatting from CSS, to be displayed correctly. To make the process as uncomplicated for the user as possible, we decided to automatically import these files into our program. To do this, the XHTML document is scanned line by line for import statements. Using string manipulation, the exact places where these statements point to are then determined. Most of the times these statements specify a location relative to that of the selected file, which means that the absolute position in the file system has to be determined, to import the dependency. After a quick check, the paths are added to the list of documents

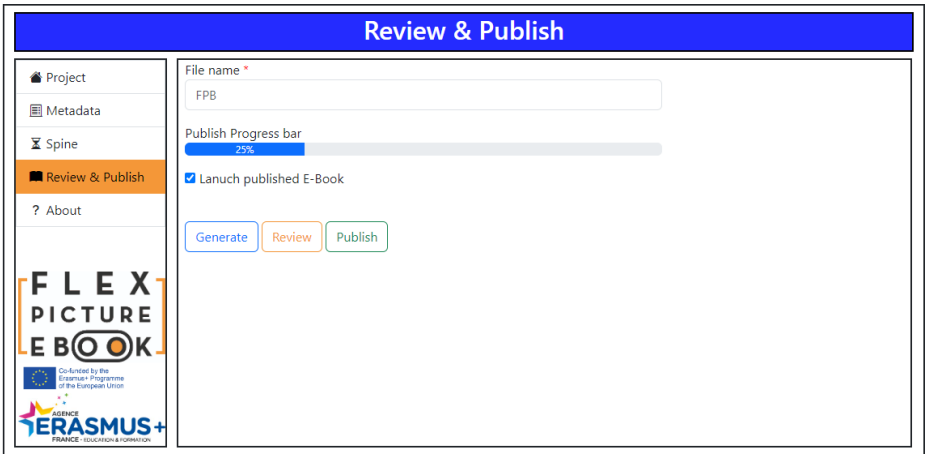

Fig. 5. Review and Publish Screen

that is packaged into the EPUB. If multiple files point to the same dependency (meaning that a file has been reused in multiple XHTMLs), the system only imports it once. In case a dependency was not found, the user is prompted to manually select it.

Before the EPUB is created, all of these documents are copied into their corresponding folders. To make everything work properly, the import statements of the XHTMLs need to point to these new locations, which our system solves by changing the old paths, with the new relative ones. Because every EPUB has the same file structure, the creation of the relative paths can be standardized.

There is a number of files that our system generates, such as the menu that is shown at the start of the EPUB, which are dynamically generated from templates. These templates are then filled according to metadata specified in the front end. This allows us to quickly create FPBs in multiple pre-specified languages. One of the issues that are still at play, is the fact that for every language the template has to be manually translated, which is a huge amount of work for us. While most files are language independent, something like the settings menus still requires the names of the buttons to be translated. Most other templates are generated completely dynamically, but the menus are (mostly) pre-written XHTML files, where some content is changed according to the specifications of the user. This is due to the comparatively large size of the menu files, which made this approach more efficient during production. However, this can be a bit of a hurdle in terms of translation, because the text content is entangled in the HTML code. One solution that will be explored, is creating the menus from language agnostic templates and reading the text from files that only contain the names of the items, thus allowing these files to be translated by an automatic translation service like DeepL[5]. These would still have to be checked by natives of that language, but could cut down on a large amount of time and potential costs.

The system is built in such a way, that if every required field was filled in, then the program exports a working EPUB, that can get validated without issues. The output of the program contains two parts: The EPUB and a folder containing all of the raw files, allowing the user to browse them and make sure they contain the right information.
As stated before, the EPUB itself is a ZIP containing all of these documents, with the file extension changed to the *.epub* ending. However, there are some more things that could potentially trip up our program: First, the format requires the ZIP to be in a particular order, namely that the .mimetype file (which tells the eReader that the folder in question is indeed an EPUB), is at the first position. Secondly, the compression inherent in a ZIP File cannot be applied equally on all documents, specifically the .mimetype file has to be left uncompressed and unencrypted[7]. Ordering the elements inside of a ZIP, or compressing elements non-uniformly can be quite challenging without specialized software, so we made sure that the rendering process was as automated as possible.

4 Future Work

Our software should lessen the workload and required technical knowledge to create Flex eBooks and lower the barrier of entry to such a degree that the format becomes more widespread. One of the ways this process could be further simplified, is through the creation of a tool that allows illustrators to export directly to the required format and simplify the process of manually abstracting images. In the next steps, this tool will be implemented as a plugin for Adobe Illustrator, which hopefully makes the task of image-creation easier and more accessible.

Acknowledgments. Flex Picture eBook has received funding from the European Commission's and the Agency Erasmus+ France through the action KA220-SCH Cooperation partnerships in school education. Grant Agreement No.: 2022-1-FR01-KA220-SCH-000088072.

References

1. Daisy Consortium: Epub (2024), https://daisy.org/activities/standards/epub/
2. Electron: Electon framework (2024), https://www.electronjs.org/docs/latest/
3. Google: Google docs (2024), https://docs.google.com/
4. pagina: Epub-checker (2024), https://pagina.gmbh/startseite/leistungen/publishing-softwareloesungen/epub-checker/
5. SE, D.: Deepl (2024), https://www.deepl.com/de/translator
6. Sigil: Sigil - epub editor (2024), https://sigil-ebook.com/
7. W3C: Epub zip file requirements (2023), https://www.w3.org/TR/epub-33/#sec-zip-container-zipreqs
8. W3C EPUB 3 Working Group: Epub 3.3 (2023), https://www.w3.org/TR/epub-33