Document Infrastructure for Augmented Reading

Lachlan Kermode
Brown University

Will Crichton
Brown University

1 Introduction

> *The conclusion we reach is not that production, distribution, exchange and consumption are identical, but that they all form the members of a totality, distinctions within a unity.*
>
> Karl Marx, *Economic Manuscripts*

Most people write documents in Microsoft Word or Google Docs, export them as PDFs, and read them using an application such as Adobe Acrobat or Preview.app. This suite of document technologies—this *document infrastructure*—is, however, very limiting for the types of reading augmentations found in HCI research. For example:

1. Some augmentations provide or require contextual information for elements like symbols [1] and citations [2]. Recovering or representing the definition-reference structure of a PDF document is difficult, necessitating the use of unreliable heuristics or machine learning.
2. Some augmentations involve live simulation, such as explorable explanations [3]. These documents can neither be authored in a traditional editor nor implemented as a PDF. Documents with live simulation are usually implemented for the browser and authored manually or via languages like Living Papers [4]. As websites, these documents cannot be shared in a self-contained file format.
3. Every augmentation must be implemented in a standalone reading system, meaning that augmentations cannot be easily composed or ported between systems. For example, Semantic Reader [2], Liquid Text [5], and ar5iv all provide their own walled gardens, making interoperability and migration a pain.

Our thesis is that *the augmented reading community needs better infrastructure.* With better infrastructure, research would be easier to implement, easier to test and evaluate, and easier to deploy to real users. Such infrastructure would not likely fit neatly into a single HCI paper, but would require sustained and coordinated investment across the gamut of document technologies.

Specifically, as shown in Figure 1, document infrastructure consist of five principle classes or components:

1. **Editors**, the systems for authoring a document,
2. **Languages**, the source syntaxes of a document,
3. **Compilers**, the translators from a language to a format,
4. **Formats**, the sharable/renderable representations of a document, and
5. **Readers**, the systems for viewing a formatted document.

The goal of this paper is to start a discussion about this infrastructure within the augmented reading community. We will discuss two existing technologies, the EPUB format and the Typst language, which we believe hold great promise in the task of building better document infrastructure. Along the way, we will discuss two new systems we are building, the Bene EPUB reader and the Rheo Typst compiler, which partially address issues in adopting EPUB/Typst as infrastructure for research on augmented reading.

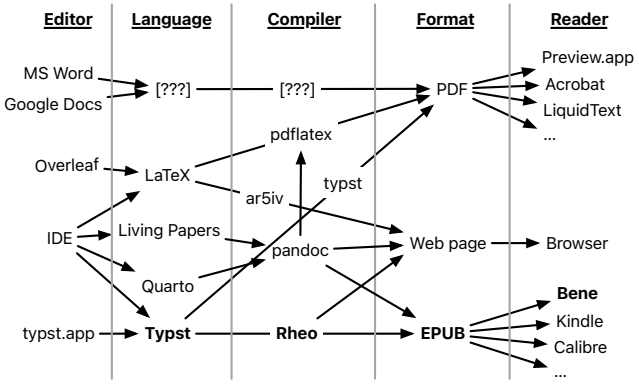

Figure 1: A schematic overview of document infrastructure, following the flow of documents from the writer (left) to the reader (right). This paper is drawing attention to the bolded technologies.

2 EPUB: A Better Document Format

As shown in Figure 1, most document compilers either generate a PDF or a standalone HTML webpage with CSS, JS, and static assets. The Portable Document Format's great advantage is, true to its name, portability. Portability means a document is *encapsulated* (wrapped into a single file) and *reproducible* (renders the same across all machines). PDFs, however, lack structure and cannot express most forms of interactivity, and attempts to extend PDF with structure/interaction are not sufficient for our community's needs [6].

The HTML format's great advantage is its flexibility. A document can be dynamically laid out using different viewports and fonts, and it can support a wide variety of interactions. However, HTML websites are not nearly as portable: they cannot always be easily packaged as a single file, they require a heavy-weight appliaction in the browser to be reliably rendered, and dynamic layout can lead to unpredictable ugliness.

We believe that the best candidate format to combine the portability of PDFs with the flexibility of HTML is the EPUB format. An EPUB encapsulates an XHTML-based document into a semi-structured ZIP file [7]. An EPUB can contain multiple "renditions" of documents which are either fixed-layout (for reliability) or fluid-layout (for flexibility). EPUB

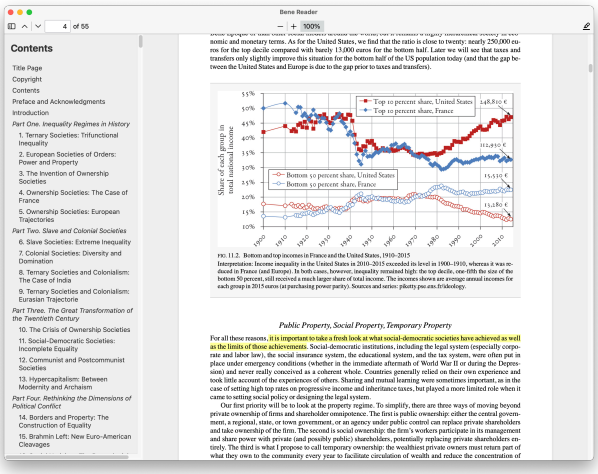

Figure 2: A screenshot of the Bene EPUB reader, showing Piketty's *Capital and Ideology*.

is widely recognized as the primary format for e-books, and it is stewarded and standardized by the non-profit W3C organization.

2.1 Bene: A Better EPUB Reader

Today, EPUB is a distinctly secondary format to PDF, generated largely to satisfy Kindle users. This is unfortunate. For EPUB-oriented reading augmentations to have an impact, people need to be comfortable using EPUB as their primary document format. We believe part of the issue is a lack of an EPUB reader that has the following attributes:

- *Lightweight:* opening an EPUB should take milliseconds. EPUBs should not have to be registered in some local persistent "library."
- *Configurable:* a reader can quickly and easily change important reading parameters, such as document width, font face, font size, and background color.
- *Extensible:* a reader should be able to add plugins which enhance the reading experience for note-taking, annotations, or other such augmentations.

Existing EPUB readers like Calibre, Apple Books, and Adobe Digital Editions do not check all of these boxes. We are therefore building Bene [8], as shown in Figure 2. Bene is lightweight by being implemented in Rust and carefully architected to optimize for speed (e.g. loading renditions in parallel, decompressing files from the EPUB only as-needed) and size (full application binary is 13 MB, vs. e.g. 836 MB for Calibre). Bene is somewhat configurable with respect to document size and font size. We are starting to work on an extensible architecture for Bene, akin to JupyterLab's plugins, for interface augmentations.

In the short term, our goal for Bene is to be our daily driver for reading EPUBs, making it pleasant to use. In the long term, our goal is for Bene to facilitate research via its extension points.

3 Typst: A Better Document Language

The most widely adopted document language to produce PDF is LaTeX, which is the de facto standard for scientific and professional typesetting. When mathematical notation or a complex layout system is not required, however, many authors prefer Microsoft Word, Google Docs, or a similar WYSIWYG editor on account of LaTeX's complex toolchain and un-ergonomic syntax. LaTeX also relies on a macro system rather than offering an in-built computational model, which makes extension and complex programming in the language awkward.

Markdown, or similar markup languages such as AsciiDoc, excel at concisely expressing documents with a relatively simple structure consisting of headers and paragraphs, such as blog posts. However, plain CommonMark Markdown lacks many features necessary for more complex documents:

- Markdown lacks syntax for custom elements. Authors either fall back to HTML, use bespoke syntax for individual elements as with Github-Flavored Markdown, or use a generic syntax for custom elements such as the triple-colon block provided by AsciiDoc and adopted by Pandoc.
- Markdown lacks support for templating or macros. Authors either layer on a separate templating language (e.g., Jinja2 or Handlebars), or push templating later in the document compilation pipeline (e.g., Pandoc's HTML templates).
- Markdown lacks support for interactive documents. Authors either use an extended Markdown integrated with frameworks like React such as MDX, or attach custom Javascript to Markdown-generated pages elsewhere in the pipeline.
- Markdown has no inherent means of compiling to PDF, or articulating backend-specific rendering directives. Authors must use a tool like Pandoc which converts Markdown to LaTeX and compiles it via pdflatex.

While every individual limitation of Markdown has some means of overcoming it in some system, the aggregate effect is that (a) any given system does not have a solution to every problem, and (b) the solutions appear more as patched-in features than a coherent design.

For instance, Living Papers [4] builds on Pandoc-flavored Markdown to support reactive documents which can compile to PDF or a webpage. The choice to build on Pandoc is completely sensible, as Pandoc is still the most robust tool for compiling documents to both PDF and web targets. However, we believe that the resulting user experience suffers from the complexity of this incrementally evolved language. For example, Figure 3a shows a simple Living Papers document. We argue that users ought not have to learn YAML, Markdown, Pandoc-extended Markdown, and LaTeX in order to write realistic documents.

```
---
title: "Minimal Living Papers snippet"
output: {html: true, latex: true}
---

# Introduction
Living Papers creates documents @sample-ref.

::: figure {#plot .center}
![Sample plot](plot.png)
| Interactive visualization with reactive binding.
:::

Figure @plot shows our results. For complex layouts, use
LaTeX:

\begin{figure*}[t]
  \centering\includegraphics[width=0.8\textwidth]{wide.pdf}
\end{figure*}
```

(a) Living Papers.

```
#set document(title: "Minimal Typst Demo")
#title()

= Introduction
Typst creates documents @sample-ref.

#figure(
  image("plot.png", width: 80%),
  caption: [Interactive visualization with reactive
binding.],
) <plot>

Figure @plot shows our results. For complex layouts, use
Typst:

#if target() == "pdf" {
  place(top + center, scope: "parent", float: true)[
    #image("wide.pdf", width: 80%)
  ]
}
```

(b) Typst.

Figure 3: An example of a short document with a plain figure and a two-column figure, written in Living Papers and Typst.

We consider the most promising alternative technology to be Typst. Typst combines the expressive layouts and PDF support of LaTeX with the ergonomics of Markdown. It supports many modern programming language features such as variables, imports, and stateful and declarative computation. Typst has a concrete and concise syntax for footnotes and citations, and as shown in Figure 3a, an elegant syntax that can express visual constructs such as layout placement, tables, figures, colors, and mathematical formulas. As Typst supports custom functions and imports, different augmentations could be written as libraries in a unified ecosystem and be incrementally adopted in that manner. Typst also has a notion of a compilation `target` built into the language, meaning that augmentations can render differently according to the capabilities of a document's reader.

3.1 Rheo: A Better Document Compiler

The main obstable to adopting Typst as a platform for augmented reading research is Typst's still-experimental support for generating web documents. As of early 2026, Typst has experimental support for exporting HTML, which supports many of the essential features for academic documents such as text decoration, headings, hyperlinks, footnotes, and citations. Typst presently has no support for generating reactive JavaScript programs like Living Papers, or generating EPUBs.

To address these shortcomings, we built Rheo [9], a toolchain around Typst to provide preliminary support for key web-related features. Most notably, in our current prototype, Rheo can collate multiple Typst documents to produce a single EPUB. More generally, Rheo enables concurrent multi-format compilation, making it plausible to compile a Typst document to PDF, EPUB, and HTML simultaneously (in the upstream Typst CLI, PDF and HTML must be generated with two separate commands). Rheo also adds several other quality-of-life features for organizing Typst projects such as relative linking between content separated across multiple Typst files[1] and static site generation features not yet supported in the upstream.

Rheo is designed as a Typst toolchain that creates a better foundation for augmented reading research. We envision that new augmentations can be implemented as toolchain modules, much as Rheo makes it possible to produce a new format, EPUB, from Typst source files. Augmentations in the domain of the document language, such as a new function or keyword in addition to Typst, can also be implemented as Rheo compiler modules. In the following two sections, we outline how two such augmentations, hyperlink previews and interactive figures, can be implemented using Rheo and Bene.

4 Conclusion

Our document infrastructure is aging. As of this year, PDF turns 33, TeX turns 48, and Markdown just graduated college at 22. We believe now is a good time to revisit the technical foundations of our research. We hope that this paper can spark a productive discussion about how the augmented reading community can learn from and contribute to the next generation of document infrastructure.

[1] There is currently a proposal to make multi-file output plausible in the upstream CLI, which would make something like this feature available in Typst by default.

Bibliography

[1] A. Head *et al.*, "Augmenting Scientific Papers with Just-in-Time, Position-Sensitive Definitions of Terms and Symbols," in *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, in CHI '21. Yokohama, Japan: Association for Computing Machinery, 2021. doi: 10.1145/3411764.3445648.

[2] K. Lo *et al.*, "The Semantic Reader Project," *Communications of the ACM*, vol. 67, no. 10, pp. 50–61, Oct. 2024, doi: 10.1145/3659096.

[3] B. Victor, "Explorable Explanations." 2011.

[4] J. Heer, M. Conlen, V. Devireddy, T. Nguyen, and J. Horowitz, "Living Papers: A Language Toolkit for Augmented Scholarly Communication," in *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, in UIST '23. 2023. doi: 10.1145/3586183.3606791.

[5] C. S. Tashman and W. K. Edwards, "LiquidText: A Flexible, Multitouch Environment to Support Active Reading," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, Vancouver BC Canada: ACM, May 2011, pp. 3285–3294. doi: 10.1145/1978942.1979430.

[6] W. Crichton, "Portable EPUBs." Accessed: Feb. 02, 2026. [Online]. Available: https://willcrichton.net/notes/portable-epubs/

[7] "EPUB 3.3." [Online]. Available: https://www.w3.org/TR/epub-33/

[8] "Bene: A Lightweight EPUB Reader." Accessed: Jan. 24, 2026. [Online]. Available: https://github.com/nota-lang/bene

[9] "Freecomputinglab/Rheo." Jan. 2026.