Single Source Publishing Case Study: Ketida (Books)

An invesitgation of how Ketida achieves Single Source Publishing for books.

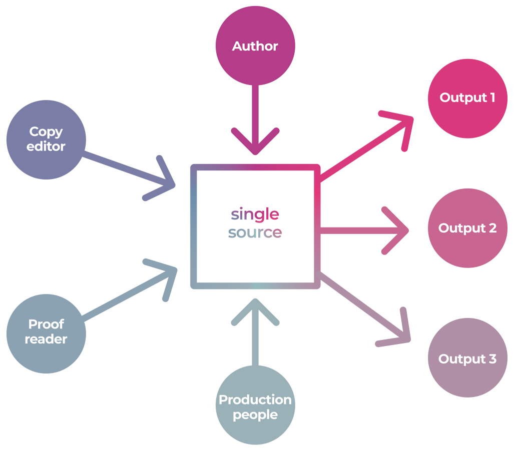
Published: Sep 13, 2021
Author: Adam Hyde
Illustrations: Henrik van Leeuwen
PDF Design: Harshna Haswani
Rendered from HTML using Paged.js

Abstract

This article is a case study of a Single Source Publishing platform for books—Ketida (built by Coko). The content is brief and by no means a complete overview of Ketidafeatures. Some demos have been prepared—when the demo icon appears in the text please click on the iconto see the pre-recorded demos.

https://coko.foundation/images/uploads/editoriaDemo.webm

Ketida is a web-based print and ebook book production platform that enables concurrent workflows and 'push button publishing' via Single Source Publishing (SSP) architecture.

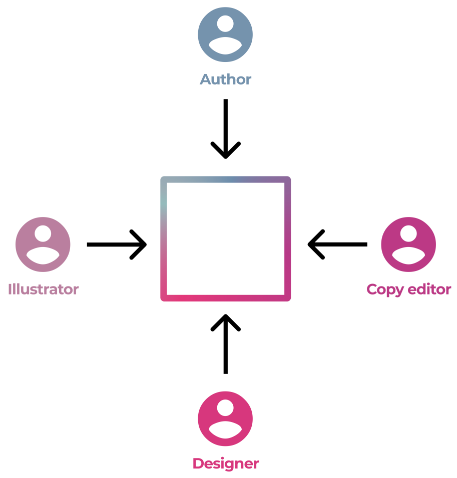


There are three core design principles:

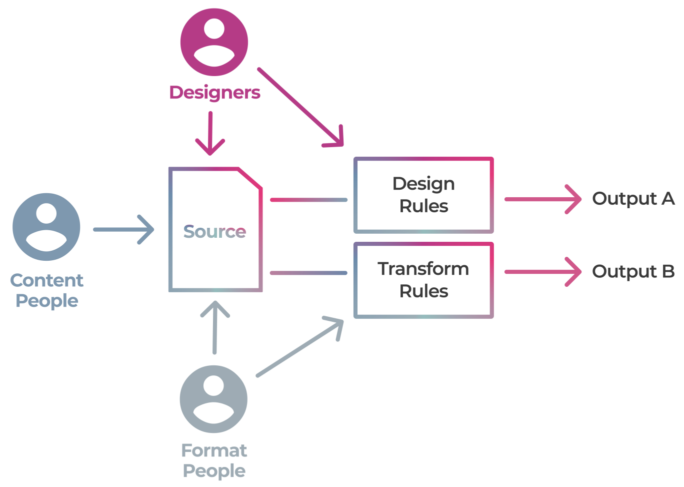
- content components—the book is treated as a collection of distinct content components (chapters, copyright pages, back matter materials etc—we will refer to these all as components for the rest of this article.

Ch1
Ch2
Ch3
Ch4

- single source—all team members (authors, copy editors, illustrators, designers etc) share the same document source file.



- rule-based design—design rules (for EPUB, PDF etc) are applied to, but are independent of, the shared source file format.

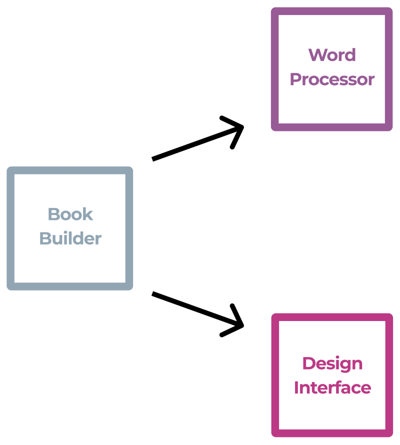


The following explores how these three design principles manifest in the three main Ketida user interfaces within a SSP architecture, and discusses the opportunities this presents for workflow optimization.

User Interfaces

There are three main interfaces that are shared by all team members:

- The Book Builder—used for building and managing the structure of the book, understanding the status of the book, and accessing all content
- The word processor—used for creating and improving content in the book
- Design interfaces—used for designing the book



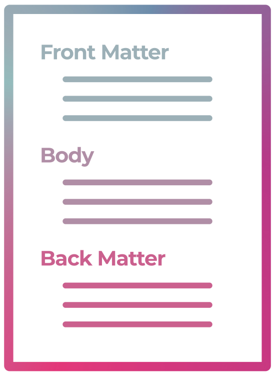
The Book Builder

The Book Builder is the interface the team collaboratively uses to manage the top level book structure, understand state, and access all content.

From the Book Builder, each team member can:

- manage the book structure
- understand the current structure of the book at a glance
- understand the current state of any chapter/component in an instant
- understand immediately the tasks they have to perform (via the state markers)
- immediately access the chapter or component they need to work on

Ketida breaks a book into individual components and displays these on the Book Builder grouped into front matter, body, and back matter divisions.



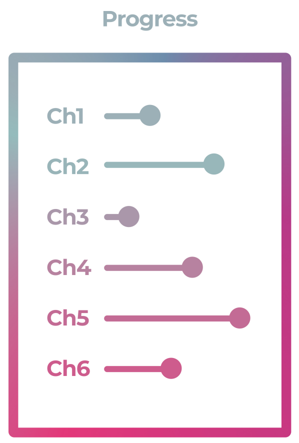
Components can be reordered by drag and drop. Updates to the book structure are updated in real time so all team members see changes immediately.

https://coko.foundation/images/uploads/toc.gif

Component-level workflow states (eg ‘for review’, ‘proofing’ etc) are also managed from the Book Builder. Each component is marked with its own state, independently of the other components. Users with the correct permissions can change a component state (move it from one status to another). A component state change is also updated in real time for all users.

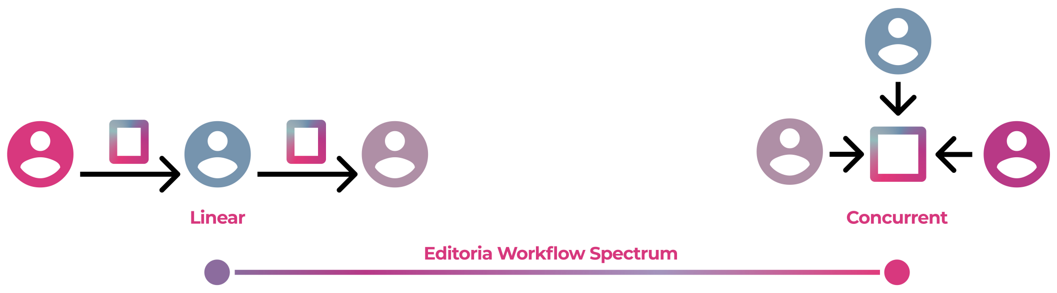
https://coko.foundation/images/uploads/concurentstates.gif

Since all components have their own individual state, workflow can be optimized by enabling each user to advance through the production process at their own pace, allowing for many different types of tasks to occur in parallel across the entire book.



Components states are tied to configurable roles and permissions. These role permissions give very granular control per state, per chapter. It is possible, for example, to allow an author to edit a chapter at a particular stage of the workflow (e.g. 'author revision') while at the same time preventing the author from turning track changes off. It is also possible to enable commenting-only for specific roles when a chapter is in a specific stage etc.

Workflow states and permissions give the publisher full control over how linear or concurrent they wish their workflow to be.



From the Book Builder, a user can click 'edit' and Ketida's word processor will open that component for content creation or editing.

On opening a component to edit, the component becomes locked. If a component is locked, other team members can still view the content in read-only mode. We have found many publishers prefer component-level locks to simultaneous realtime editing of (for example) chapters.

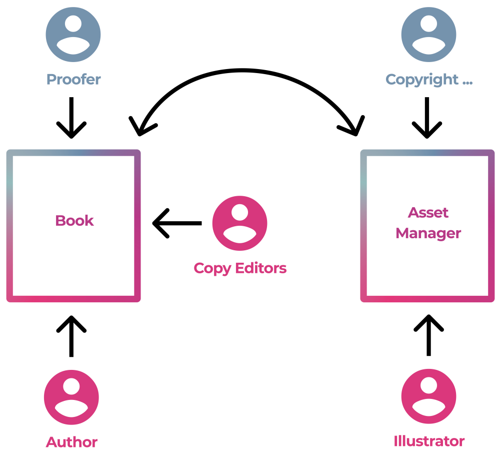
Content Creation and Editing

Content is created/edited via a web-based word processor (a purpose built software module named 'Wax' after the Greek wax tablets).

Wax is itself very configurable and extensible via plugins. By default, Wax supports all the general text functions you would expect from a word processor, including track changes, math, tables, special character support, threaded comments, images, footnotes, etc. Wax can also integrate with third-party services easily (such as automated processes for structuring text, or Grammarly for proofing etc). Wax supports simultaneous realtime editing but it is not implemented in Ketida at present (see above).

https://coko.foundation/images/uploads/Peek-2021-09-14-09-56.gif

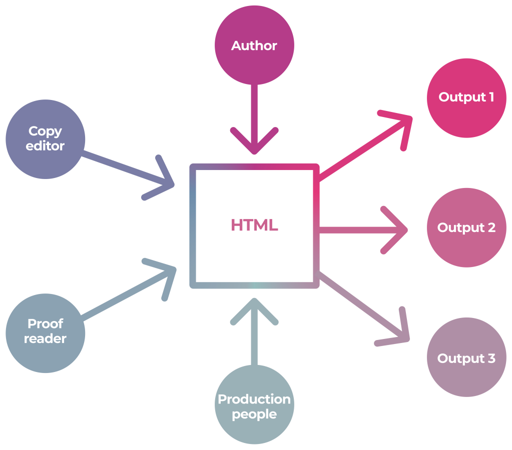
All images for a component are managed by Ketida's Asset Manager which can be accessed either from Wax (within a component) or from the Book Builder. The Asset Manager allows for assets (mostly images) to have their own workflow independent of other content. This allows the artwork team (if there is one) to work in parallel to other team members (copy editors, authors etc).



When editing or creating text, all document semantics (headings etc) must be applied using 'named styles' which means the underlying document structure is as it appears to the eye (WYSIWYM—'what you see is what you mean'). All styles are pre-determined and set by the publisher. It is not possible with Wax to make a paragraph look like a heading (via applying arbitrary fonts and size choices within Wax) without actually using the appropriate publisher-pre-defined heading style.

When working in a single source platform such as Ketida, ensuring the underlying syntax of the document matches how it appears is very important. Designers, for example, cannot apply their design if they do not have known and reliable document semantics. Given that authors, copy editors, proofers, designers etc all share the same source, the underlying 'display semantics' must correlate to explicit, known markup throughout the entire production process.

The document format is obviously an important design decision when choosing or building a word processor that supports the above features in a Single Source Publishing system. Ketida's underlying document storage format is HTML.



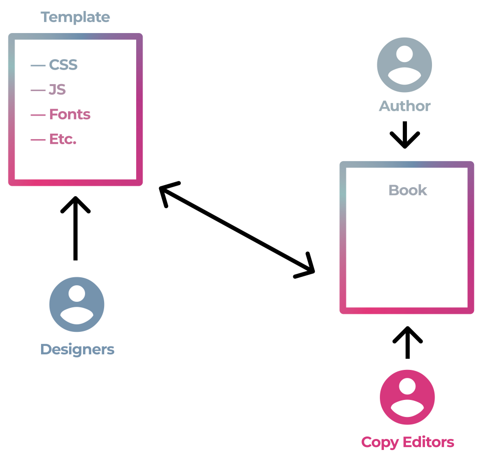
HTML has many advantages as a storage format in this context but first and foremost are:

- HTML has a near universal set of established common document semantics AND can carry bespoke semantics of any publisher easily
- HTML can carry a lot of additional structural information via class information
- HTML can be progressively structured without breaking.

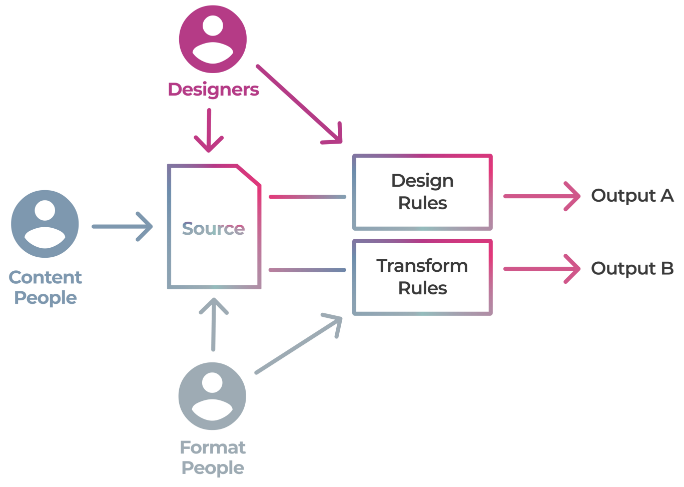
In addition to the above advantages for word processing, HTML also offers book designers opportunities in a Single Source Publishing system.

Design Interfaces

In Ketida, design for EPUB and print-ready PDF is managed via editable rules that are stored together with various other design assets (eg fonts, JavaScript snippets) as templates.



The design system combines the semantically-controlled HTML content source (produced by the word processor Wax) with the predefined design rules (templates) to produce EPUB and PDF.



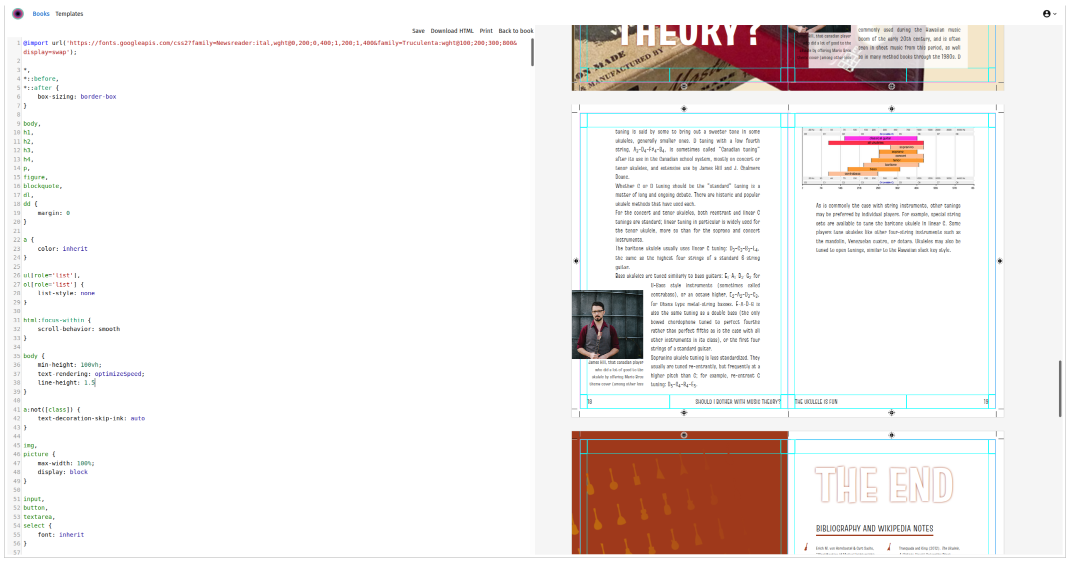
It is the role of the designer to create CSS that targets the (known) output semantics from Wax, together with any format-specific controls (eg running heads, widows and orphans, etc), and produce beautiful looking print and ebooks. Designers do this by adding bespoke classes via Wax if necessary, as well as editing the design templates live.

Designers can add custom classes (inline and block) to the source via the Wax editor for design purposes. All custom class names are carried through unaltered in the HTML to allow ease of targeting via CSS. The CSS design rules for print or ebook can be edited at anytime to improve the design.

PDF output is handled by PagedJS (although it is also possible to export to InDesign formats). PagedJS is a typesetting system that follows the W3C PagedMedia pagination control standards to convert CSS and HTML into PDF.

Effectively, PagedJS extends CSS to include additional controls for designing all elements of a print book.

PagedJS CSS can be edited live in the paged.js design interface.



```css
@import url('https://fonts.googleapis.com/css2?family=Newsreader:ital,wght@0,200;0,400;1,200;1,400&family=Truculenta:wght@100;200;300;800&
display=swap');

*,
*::before,
*::after {
    box-sizing: border-box
}

body,
h1,
h2,
h3,
h4,
p,
figure,
blockquote,
dl,
dd {
    margin: 0
}

a {
    color: inherit
}

ul[role='list'],
ol[role='list'] {
    list-style: none
}

html:focus-within {
    scroll-behavior: smooth
}

body {
    min-height: 100vh;
    text-rendering: optimizeSpeed;
    line-height: 1.5
}

a:not([class]) {
    text-decoration-skip-ink: auto
}

img,
picture {
    max-width: 100%;
    display: block
}

input,
button,
textarea,
select {
    font: inherit
}
```

Screenshot of PagedJS Design interface

Rendering of outputs (PDF/EPUB) can occur at any time in the workflow and can be done by anyone, permissions pending. This means team members can see early galleys at the push of a button. It also means that a designer can continue designing a book and render EPUB and print-ready PDF (with the actual content in place) while the book is still being written.

Demo *https://coko.foundation/images/uploads/Peek-2021-09-14-10-08.gif*

Single Source Publishing and Ketida

Ketida is a complete single source publishing solution. The platform is built for small and large enterprises with a scalable modular microservices architecture. Almost every part of the system is replaceable, extensible and customisable—allowing for an enormous scope of workflows.

Concurrency is built into the core of the Ketida architecture. Publishers can benefit greatly from leveraging Ketida's concurrent SSP framework by optimising their workflow in innumerable ways. Some examples include (taken from the earlier article on SSP):

- Art logs and image research starting as soon as FPO ('for placement only') images are placed
- Alt text being created as images are being added

- Indexing could start as the text is completed (inserting tags) before the proof stage
- Proofing being finished on one part of the book before the rest is finished
- Concurrent page rendering enables author to correct or fit content as they write it
- Testing the content with live customers as the rest of the content is being finished (O'Reilly does this, and it's a way to do A/B testing for learning objects and assessment items in education. It's a way to improve product-market fit).
- Multiple authors working on the same text at the same time
- Designers working on design of the actual content as a work is being created
- Copy editors working on content immediately it is completed by an author
- Illustrators placing images in a work as it is being authored
- Production editors understanding exactly what has and hasn't been done at any time
- Developmental editors and authors working on the same text at the same time

Note: this article is only a partial list of Ketida features, focusing only on on some core system design decisions that lead to, or are a consequence of, Single Source Publishing principles.

Thanks to Henrik van Leeuwen for illustrations and Raewyn Whyte for copy editing.