Kotahi: a new approach to JATS production

A new innovation from Coko - push button JATS production.

Published: May 5, 2022

Author(s): Adam Hyde , Ben Whitmore (Coko) , Dan Visel (Coko)

Illustrations: Henrik van Leeuwen

PDF Design: Harshna Haswani

Rendered from HTML using Paged.js

An introduction to Kotahi

Kotahi is a free, open-source system for scholarly publishing, designed to support a wide variety of publishing use cases including journals, micropubs, preprints, PRC (Publish, Review, Curate), weblabs, and more. Kotahi supports multiple workflows for each of these use cases, with a key feature being the single-source publication to multiple formats. Once a document is in Kotahi, it can be exported to JATS (Journal Article Tag Suite), PDF, or HTML with the Kotahi source as the single source of truth.

Not all workflows require every one of these output formats – it's possible to use Kotahi for evaluating preprints residing on external servers, for example, with no need to regenerate their PDFs – but an increasing number of publishing projects aim to generate multiple formats, and Kotahi is designed to make it easy for teams with minimal format-specific knowledge to publish to any of those formats. Kotahi also versions documents; if a new version of a document is created, re-exporting in all formats is simple.

Kotahi is in active development, with a growing number of users. It is available for installation from the Coko Gitlab).

Features

Kotahi's main features include:

- Multiple workflows and use cases: Kotahi supports multiple workflows for PRC (Publish, Review, Curate) and the submission, peer review, and publishing of everything from preprints, micropubs, journals, and books to exciting new types of publishing.
- Multiple review models: Kotahi enables publishers to use a number of different review models: open, blind, or double blind. Reviewers can have individual reviews or collaborate on a shared review. Review forms are customizable from within the browser.
- Document editing: Users can author and edit articles in the browser, using Wax (https://waxjs.net/), Coko's full-featured web-based word processor.
- Real-time notifications and chat: Kotahi is built to get the best from web collaboration and supports real-time notifications, chat, synchronized updates and more. Live chat (both text and video) functionality can be used to communicate with authors or collaborate as an editorial team.
- Metadata: The in-browser form builder can be used to design submission forms to capture and manage metadata specific to each workflow.
- Single-Source Publishing: Kotahi supports single-source publishing (https://coko.foundation/articles/single-source-publishing.html). All

stakeholders are able to access the same manuscript during content creation (peer review, editing, etc.) and production (copy editing, semantic tagging, etc.). This obviates the need to track and control documents across multiple systems.
- Versioning: Versioning is supported so that changes to a submission can be easily compared.
- Exporting: Currently Kotahi exports to PDF, JATS, and HTML. A GraphQL API makes it possible to export Kotahi's data to other systems (such as PubMed, Crossref and FLAX) as needed.
- Integrations: Authorization integration via ORCID is available. Kotahi can register DOIs for articles via integration with Crossref. It can also publish reviews of existing articles as Hypothes.is annotations, as used by eLife for generating Transparent Review in Preprints (TRiP) on the bioRxiv server.
- Reports: Admin users of Kotahi get a dashboard that dynamically displays the progress of articles through the system.

Users

Development of Kotahi is being supported by eLife, Amnet Systems, and Aperture (an open publishing platform from the Organization for Human Brain Mapping).

This article primarily focuses on the production of JATS (https://jats.nlm.nih.gov/) in Kotahi. These capabilities have been anticipated for many years by the Coko team and the build up to the current JATS functionality has taken many years. However it is worth noting that recently Coko has collaborated with Amnet, a publishing solutions company in India, on the Wax-JATS development.

Background: Coko and Kotahi

The Coko Foundation was founded with seed funding from founder Adam Hyde's Shuttleworth Foundation fellowship. Coko produces a variety of open-source software for publishing. Kotahi is one of many open source publishing-related projects from Coko, and is built on top of several prior Coko projects.

Coko frameworks used by Kotahi

Kotahi builds on a number of different projects produced by Coko. Major components include:

- PubSweet. PubSweet is Coko's open source framework for building state-of-the-art publishing platforms. Kotahi uses this as a component library and parts of its framework. PubSweet is written in JavaScript.
- xSweet
- Wax allows marking up of documents in a way that can be easily translated into valid JATS. Wax and Wax-JATS are written in JavaScript.
- Paged.js to create high quality PDF output from any HTML content. Using instance-specific templates, this allows Kotahi to export manuscripts and metadata as print-ready PDFs. Paged.js is written in JavaScript.

In conjunction with Kotahi, integration work is also being done on another Coko project:

- FLAX (website forthcoming). FLAX is an open source publishing front end, a web presence for content produced in Kotahi and Editoria (https://editoria.community/). FLAX is in active development as a way to present content which lives in Kotahi in a web-first model. FLAX is written in JavaScript.

And two other projects at Coko, which may be integrated with Kotahi in the near future:

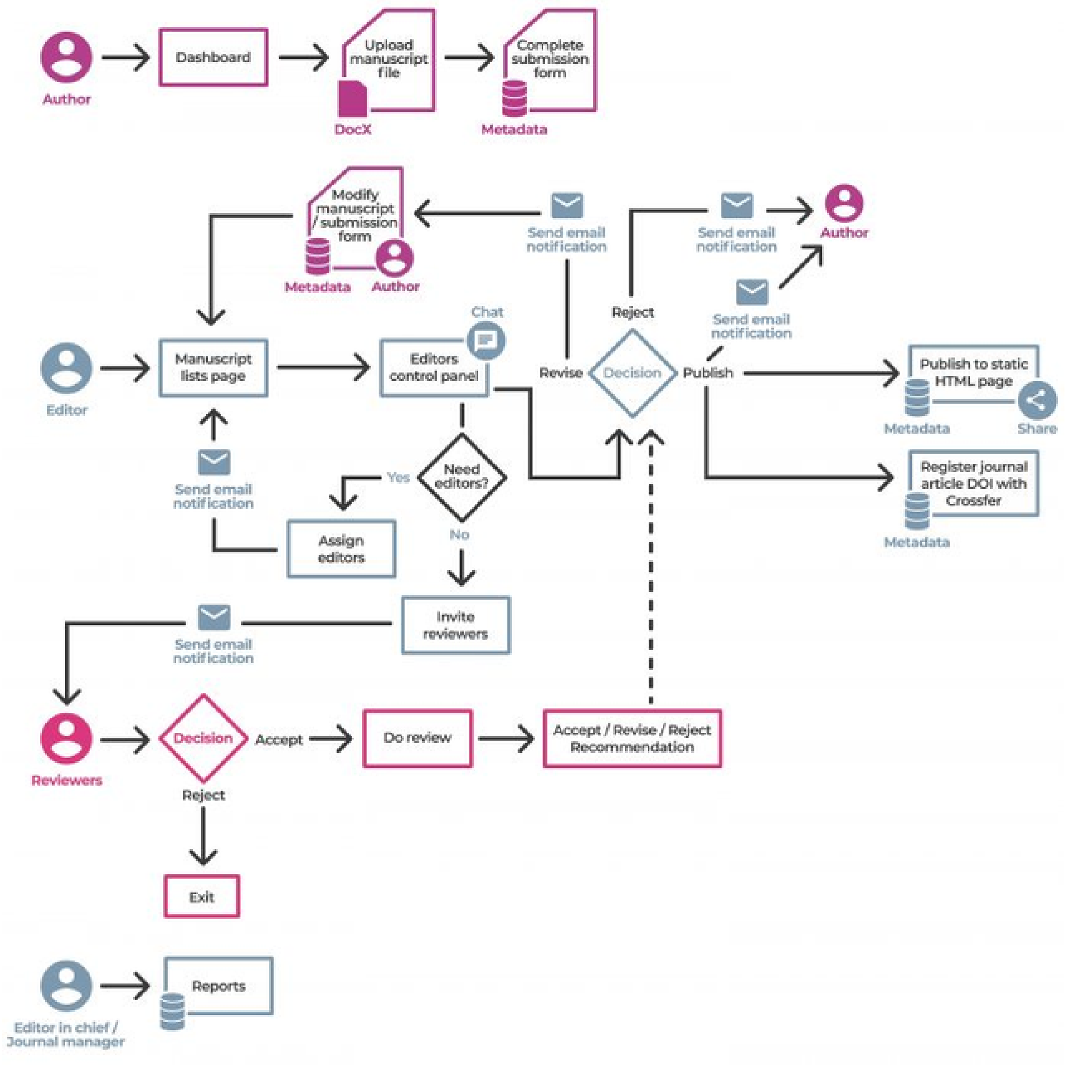
- Science Beam (https://sciencebeam.org/). An open source machine learning application to convert PDF scholarly articles to XML with high accuracy. Originally built by eLife and transferred to Coko to further develop and maintain. ScienceBeam is primarily a Python application.
- Libero Editor (website forthcoming). An open source JATS XML editor (a replacement for Texture). Originally built by eLife and transferred to Coko to further develop and maintain. Libero Editor is written in JavaScript.

While traditional workflows involve direct editing of documents inside of Kotahi, it's also possible to use Kotahi to run a PRC workflow, ingesting pre-prints from external servers. In that particular case, ScienceBeam might be used to generate JATS from PDF-based manuscripts. In this use case Libero Editor would then be used to edit the JATS produced by ScienceBeam. This same pipeline could be used for other scenarios that need to convert PDF back into structured data (eg. where LaTeX PDF output has been submitted).

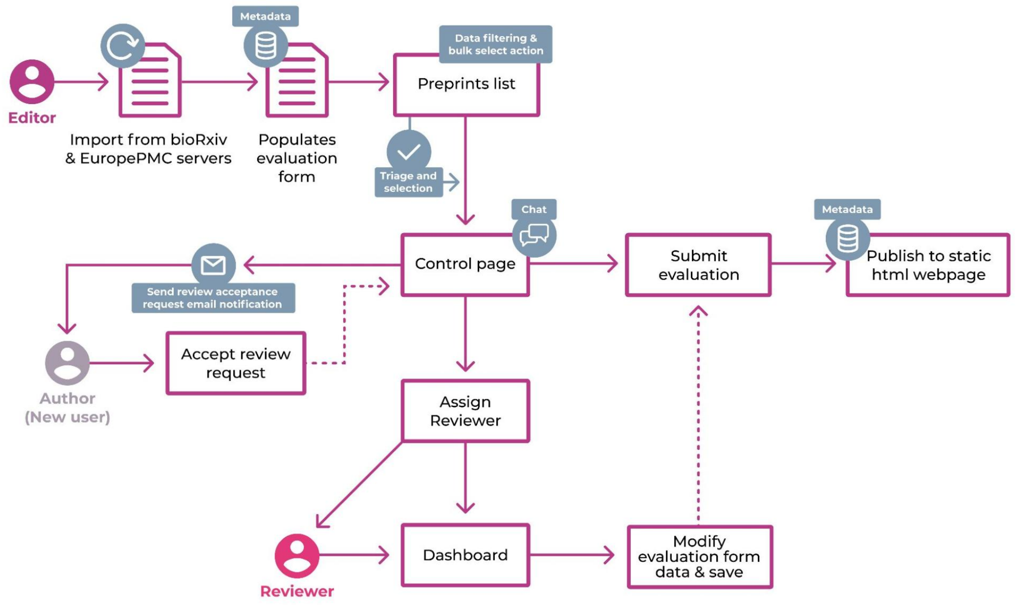
Key concepts used by Kotahi

Workflow

The Kotahi workflow is very configurable and can support workflows that range from very linear to very concurrent. Submission types are also configurable. These two features together mean that Kotahi can support a wide variety of use cases and workflow models. As an example, here's a diagram of how Kotahi is being used by the Aperture Neuro journal (Organization of Human Brain Mapping):



The following is also a workflow diagram showing Kotahi being used by a PRC organization (Publish, Review, Curate):


Editor
Import from bioRxiv & EuropePMC servers
Metadata
Populates evaluation form
Data filtering & bulk select action
Preprints list
Triage and selection
Chat
Control page
Submit evaluation
Metadata
Publish to static html webpage
Send review acceptance request email notification
Author (New user)
Accept review request
Assign Reviewer
Reviewer
Dashboard
Modify evaluation form data & save

These diagrams don't need to be explored in depth for an understanding of how JATS fits into Kotahi (see https://coko.foundation/articles/white-paper-kotahi-current-state.html), though it's useful for thinking about the ways documents travel through a system from user to user. Many different people are viewing a manuscript – in different states – and may be making changes; and every publication probably has their own, slightly different, path. Aperture, for example, is publishing to static HTML pages and PDF as a one-time event; but you can also configure Kotahi to publish multiple times synchronously or asynchronously. Most of the time we imagine JATS as being the final point in a document; but Kotahi is designed in such a way that it doesn't have to be.

HTML

Kotahi is using HTML as the source of truth. This has been core to Coko's philosophy - bringing publishing to the web.

However, we're also realistic about the ways in which authors are working: they're largely using Microsoft Word to prepare manuscripts – and editors by and large seem to resist moving away from MS Word. In this case, we need to bring the Word Documents into the web, that is, convert the docx files to HTML at submission time.

MS Word poses particular challenges: Although the content of a docx file may look nice, the underlying source is very complicated and messy. Consequently, computationally determining the correlation of author intent to underlying document markup is difficult. xSweet approaches the problem by computationally interpreting the very messy XML source of docx files and converting to well-structured and clean HTML.

The following is an excerpt taken from article written by Adam Hyde and Wendell Piez (the designers of Coko's xSweet) on this approach:
"...attempts to 'get out of Word' have tried to jump from unstructured MS Word to very structured XML formats by:

1. copying over all the data in the document and
2. interpolating structure at the same time in an attempt to 'understand' the 'intent of the author' (or a proxy) as represented by the display semantics of the document.

But if the structure does not exist in the first place, you have a problem.
xSweet's Word-to-HTML conversion retains step one (copying over all the data) and replaces step two (interpolating structure) with a process that forsakes the introduction of any structure in favor of carrying over whatever information from the original MS Word file might be useful later on, whether for programmatically understanding or manually applying document structure. The best solution of course, being a little bit of both. Hence we:

1. convert the unstructured MS Word document into an unstructured (or partially better structured) HTML document; and
2. interpret the original MS Word file and carry forward as much information as that original Word file contained for possible future use in interpreting the document structure – or representing any features of interest – while not actually structuring the document.

Interestingly, since HTML does not require you to enforce a strict document structure if you do not have it, an unstructured document flows into HTML as easily as a structured, or partially structured, document flows into it. If your aim is a well-controlled document format, such a failure to enforce strict structures is regarded as a flaw or weakness. Yet, since we do not have much or any document structure in the originating Word document, and our goal is to improve it – HTML's flexibility becomes a critical feature."

This process produces an intermediary carrier format – an information-rich, sanitized HTML format that is suitable for improvement.
A final step of the xSweet libraries further interprets the information carried over from the conversion which implies structural elements, applies that structure, and then converts the total output to a (configurable) target HTML profile.
In an HTML editor, we can then bring to bear further tools (Wax, for example) for adding structure, reviewing, commenting, and revising.

Single-Source Publishing

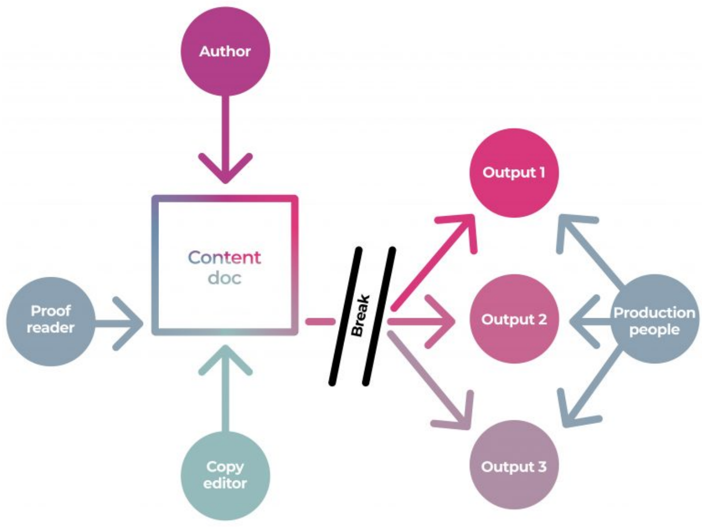
Kotahi's design is also informed by the idea of Single-Source Publishing. There’s a lot that could be said about this; where it’s relevant to Kotahi and JATS is that the Kotahi model sees JATS production as an integral part of the publishing workflow, not something that should be considered as a separate step.

Publishing has long suffered from broken workflows that slow down the time to publish and unnecessarily inflate the cost to publish. In general, this comes from a historical disconnect between content creation processes and production processes.

In many publishing environments, authors, copy editors, proofreaders and others create and improve content primarily in one tool (usually Microsoft Word). The content creation process feeds into the production process where production staff create a multitude of other formats – HTML, PDF (for print and screen), as well as XML and ebook formats.

To convert to these file formats, the content has to either be programmatically converted to various target formats via software (such as Pandoc or bespoke software), or manually converted by people using software (such as InDesign). Publishers achieve this by either slinging the content over the wall to a publishing services vendor or contractor, or they employ internal staff. The people doing these conversions belong to the general category of production people – usually programmers, designers, or format wranglers.

Most publishing workflows separate the content creation from the content production. This disconnect separates who does the work, the tools used, and, most importantly, the files worked on. The people, the tools, and the working files all change as the content jumps from content creation to content production.

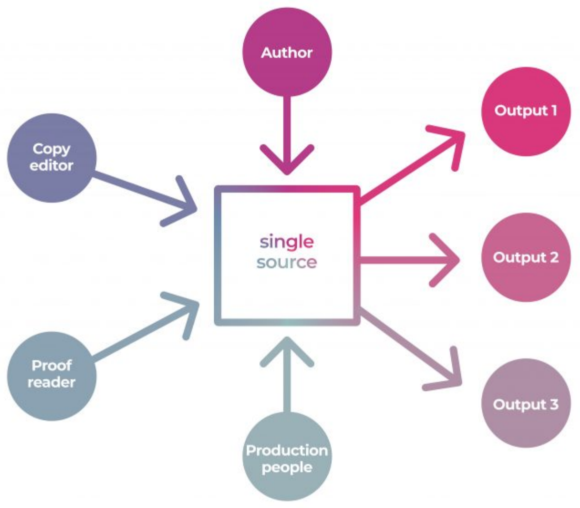


Consider a simple journal example: imagine that the production staff have discovered from proofs that figures are in the wrong place. This information must be communicated to (for example) the publishing services provider (via email, MS Word, annotated PDF, etc.). The vendor interprets the information, makes the changes in their (various) tools, and sends it back to the publisher to check. This may be iterated repeatedly.

Larger publishers have staff to manage the communication of changes, track the changes, check the changes etc. This costs time and money. If publishers don't do this well, the consequence is that more errors are introduced.

This is the problem single sourcing is meant to solve. Single sourcing is a general approach to publishing workflows that is intended to avoid disconnecting the content creation and production processes – saving time and money, and reducing errors.

Single sourcing isn't a specific solution, it is a general idea that must be intentionally designed into a publisher's workflow. Single sourcing changes how people work and often requires a different tooling. The secret really, if we zoom out to a high-level abstraction of the problem, is to work out how the content creation and production people can work in a shared environment where they all work on the same files, the same source files – hence the term 'single source.'



In a single-source environment, if a change needs to be made while the content is in production, the content people can make the change themselves. Less time and communication required, less to-and-fro, less management overhead, and fewer errors.

What this also allows, is moving from a more linear model of publishing – a manuscript goes from step A to step B to step C, each done by a separate person – to a model that allows more concurrency and collaboration, leading to more efficient document production. Efficient document production is one of the many things Kotahi sets out to achieve.

Documents in Kotahi

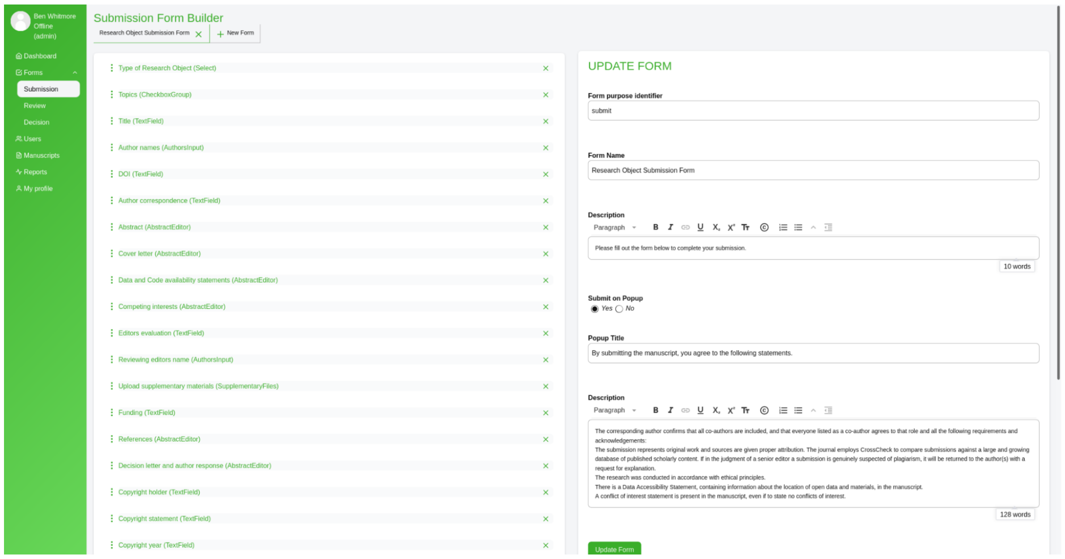
A Kotahi document is both a document and a collection of data - a network of content.

Kotahi's representation of a document can be split into two main interlocking parts:

1. a submission form which contains metadata; and
2. a manuscript editor (Wax) which contains the body of the text.

As different users edit the content or the metadata, each session is added to the document history. The most recent edit becomes the common version for all users (as per Single Source Publishing). In this way users are always working with the most recent content (single-source).

Submission forms for documents can be customized per instance in Kotahi with a drag-and-drop form builder; the forms contain the particular set of metadata that the journal uses.



Submission Form Builder

Research Object Submission Form | New Form

- Type of Research Object (Select)
- Topics (CheckboxGroup)
- Title (TextField)
- Author names (AuthorsInput)
- DOI (TextField)
- Author correspondence (TextField)
- Abstract (AbstractEditor)
- Cover letter (AbstractEditor)
- Data and Code availability statements (AbstractEditor)
- Competing interests (AbstractEditor)
- Editors evaluation (TextField)
- Reviewing editors name (AuthorsInput)
- Upload supplementary materials (SupplementaryFiles)
- Funding (TextField)
- References (AbstractEditor)
- Decision letter and author response (AbstractEditor)
- Copyright holder (TextField)
- Copyright statement (TextField)
- Copyright year (TextField)

UPDATE FORM

Form purpose identifier: submit

Form Name: Research Object Submission Form

Description: Please fill out the form below to complete your submission.

10 words

Submit on Popup: [✓ Yes] No

Popup Title: By submitting the manuscript, you agree to the following statements.

Description:
The corresponding author confirms that all co-authors are included, and that everyone listed as a co-author agrees to that role and all the following requirements and acknowledgements:
The submission represents original work and sources are given proper attribution. The journal employs CrossCheck to compare submissions against a large and growing database of published scholarly content. If in the judgment of a senior editor a submission is genuinely suspected of plagiarism, it will be returned to the author(s) with a request for explanation.
The research was conducted in accordance with ethical principles.
There is a Data Accessibility Statement, containing information about the location of open data and materials, in the manuscript.
A conflict of interest statement is present in the manuscript, even if to state no conflicts of interest.

128 words

Update Form

The submission form is completely customizable by the publishers and can be used to capture any metadata they wish. It can include, for example, the title of a document, relevant dates, author names and affiliations, and a document abstract. The form metadata could also include keywords (either freeform or from a predefined list) and file attachments. Kotahi's conception of the form is extremely abstract by design to support as wide a spectrum of use cases as possible.

The manuscript has some flexibility as well. Kotahi uses Coko's xSweet to import docx documents. For anything that's not a docx, the manuscript is treated (at this stage) as an attachment; an URL for a document that exists elsewhere on the web can also be used.

The most common workflow, however, is to import docx files into Kotahi. These go through xSweet and are returned as HTML (as discussed above) and displayed in Wax. Understanding this chain of events is critical to understanding how Kotahi enables users with no prior knowledge of XML to produce valid JATS, as will become evident later.

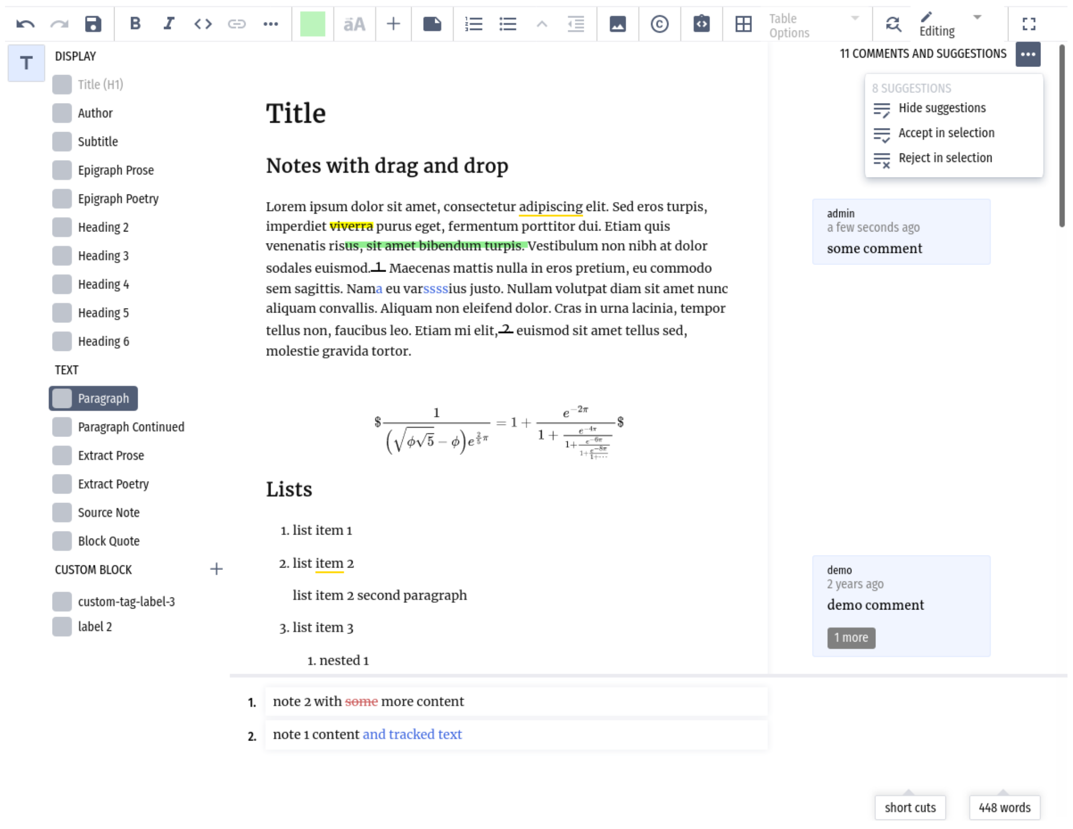
xSweet not only converts the docx file to clean HTML but also pulls metadata from the file which is inserted into the form – the primary header, for example, becomes the title of the document, though that can be overridden by the form.

When a manuscript has been imported, it can be edited by users (dependent on roles and document state). Editing happens in the Wax editor, which is a full-featured editor. Features of Wax include:

- Threaded Comments and Annotations
- Different views for different users: an author or a reviewer might see a read-only version of a Wax document which appears as the final version. Authors might be able to reply to comments but not add their own.

- Support for editing equations using MathJax: entering a $ or $$ enters math mode, where LaTeX can be entered. When math mode is exited (by typing another $ or $$), the LaTeX is converted to an SVG for display purposes; clicking it allows editing.
- Predefined (configurable) styles
- Interactive widgets (e.g. question models)
- Uploading of figures and media
- Full-featured footnotes
- Table entry and editing
- Highlighting
- Internal track changes: Additions and deletions can be tracked by user.

The following image shows the user interface for Wax.



Title

Notes with drag and drop

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed eros turpis, imperdiet viverra purus eget, fermentum porttitor dui. Etiam quis venenatis risus, sit amet bibendum turpis. Vestibulum non nibh at dolor sodales euismod.1 Maecenas mattis nulla in eros pretium, eu commodo sem sagittis. Nama eu varssssius justo. Nullam volutpat diam sit amet nunc aliquam convallis. Aliquam non eleifend dolor. Cras in urna lacinia, tempor tellus non, faucibus leo. Etiam mi elit,2 euismod sit amet tellus sed, molestie gravida tortor.

$$\$\frac{1}{\left(\sqrt{\phi\sqrt{5}}-\phi\right)e^{\frac{2}{5}\pi}}=1+\frac{e^{-2\pi}}{1+\frac{e^{-4\pi}}{1+\frac{e^{-6\pi}}{1+\frac{e^{-8\pi}}{1+\cdots}}}}\$$$

Lists

1. list item 1
2. list item 2

   list item 2 second paragraph
3. list item 3
   1. nested 1

admin
a few seconds ago
some comment

demo
2 years ago
demo comment
1 more

1. note 2 with ~~some~~ more content
2. note 1 content and tracked text

As mentioned earlier, Wax's internal format is HTML. Consequently, Kotahi can directly export PDFs using HTML with PagedMedia queries (utilizing Paged.js). Styling for Paged.js is done with stored CSS templates. Paged.js is itself a comprehensive topic and deserves a separate article.

For more information on PagedMedia and how Paged.js works please see https://pagedjs.org/

It’s important to note that Wax (as used in Kotahi) is not an ‘anything goes’ HTML editor: Wax allows only a strict predefined (and configurable) set of document semantics. Each semantic element (eg italics, or headings) are recorded in the underlying source as tags (eg. `<em>``</em>` or `<h1>``</h1>`) which align (by configuration) with the same tags produced by xSweet. While a submitted docx file might consist of a great variety of (almost) arbitrary tags, the result of xSweet’s docx-to-HTML conversion has a clearly defined, clean, tag range and structure. The output from xSweet is also conformant with the structure and tags required by Wax. Editing in Wax also ensures the underlying integrity of the underlying document source is maintained.

Making JATS-ready documents

When the staff/team are satisfied with the shape of the content, they can then prepare the material for publication using the Kotahi production interface which contains the Wax-JATS editor. The Wax-JATS editor is similar to the regular Wax editor (it is a ‘version of Wax’), but it also includes functionality specific for preparing the document to export as JATS. From this production page, the user can also download versions of the document as PDF, HTML, or JATS.

JATS in production (the Wax-JATS editor)

The Wax-JATS editor includes features related to creating JATS. The user requires no prior knowledge of JATS or XML to prepare the document for JATS export. The Wax-JATS interface simply requires the user to highlight parts of the document and choose what type of content that selection contains. This is very much a drag-and-drop and point-and-click exercise. No messy editing of XML required. Citations, for example, can be identified by selecting text and clicking on ‘citation’.

The design principle envisioned by Adam Hyde, behind the Wax-JATS editor, is not to expose the JATS document model (which is a complex set of XML tags) but to give the user simple MS Word-like tools they can use to create JATS without having any knowledge of the JATS syntax. The conversion to JATS is as automated and as simple as possible. The user doesn’t need any knowledge of what the internal format is; they only see content and visual style and use simple selection tools to identify parts of the content.

The Wax-JATS editor then marks the underlying format (HTML) in such a way that Kotahi can map these tags to JATS at export time.

For example, internally a Kotahi document might have a typical flat HTML structure such as:

```
<h1>This is a top-level header</h1>

<p>This is content</p>

<h2>This is a second-level header.</h2>

<p>This is in the second level</p>

<h3>This is a third-level header</h3>

<h2>This is a second second-level header</h2>
```

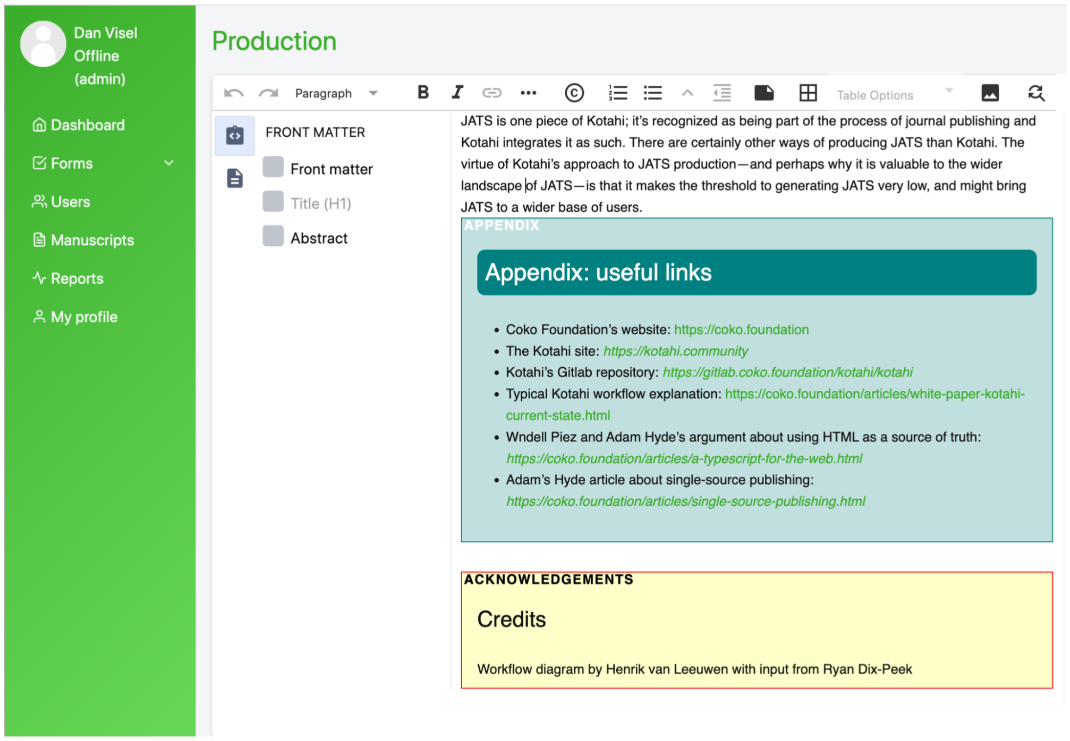
On conversion, that familiar HTML structure is carefully wrapped in JATS-style sections from the smallest header up, becoming:

```
<sec>

  <title>This is a top-level header</title>

  <p>This is content</p>

  <sec>

    <title>This is a second-level header.</title>

    <p>This is in the second level</p>

    <sec>

      <title>This is a third-level header</title>

    </sec>

  </sec>

  <sec>

    <title>This is a second second-level header</title>

  </sec>

</sec>
```

This is a more logical structure; but the user is not forced to convert the document into the nested format - that is all automated. The user simply imposes structure visually; the converter attempts to make sense of it, and exports to JATS at the push of a button.

A similar approach is currently taken with citations. The Kotahi approach is currently to wrap the selected citations in a Reference List element; every paragraph or list item that's inside a Reference List in Wax becomes a `<mixed-citation>` on export to JATS.

The following screenshots show this process in action. The first screenshot (below) displays the Wax-JATS editor with some areas of an article selected.



JATS is one piece of Kotahi; it's recognized as being part of the process of journal publishing and Kotahi integrates it as such. There are certainly other ways of producing JATS than Kotahi. The virtue of Kotahi's approach to JATS production—and perhaps why it is valuable to the wider landscape of JATS—is that it makes the threshold to generating JATS very low, and might bring JATS to a wider base of users.

APPENDIX

Appendix: useful links

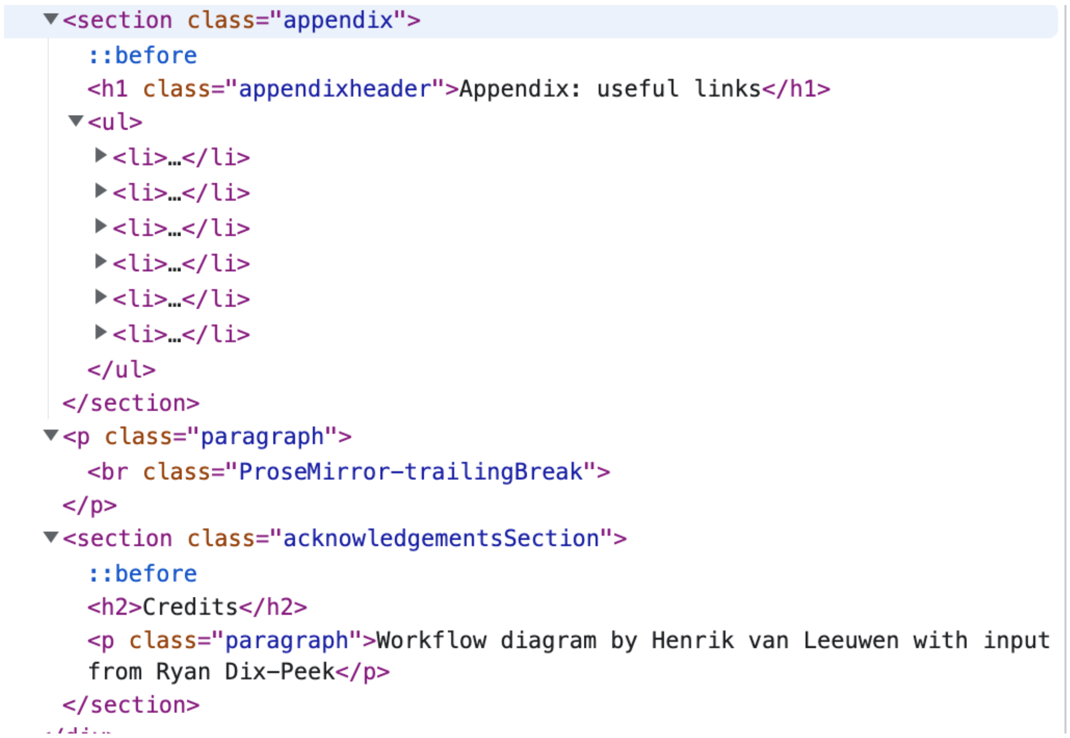
- Coko Foundation's website: https://coko.foundation
- The Kotahi site: *https://kotahi.community*
- Kotahi's Gitlab repository: *https://gitlab.coko.foundation/kotahi/kotahi*
- Typical Kotahi workflow explanation: https://coko.foundation/articles/white-paper-kotahi-current-state.html
- Wndell Piez and Adam Hyde's argument about using HTML as a source of truth: *https://coko.foundation/articles/a-typescript-for-the-web.html*
- Adam's Hyde article about single-source publishing: *https://coko.foundation/articles/single-source-publishing.html*

ACKNOWLEDGEMENTS

Credits

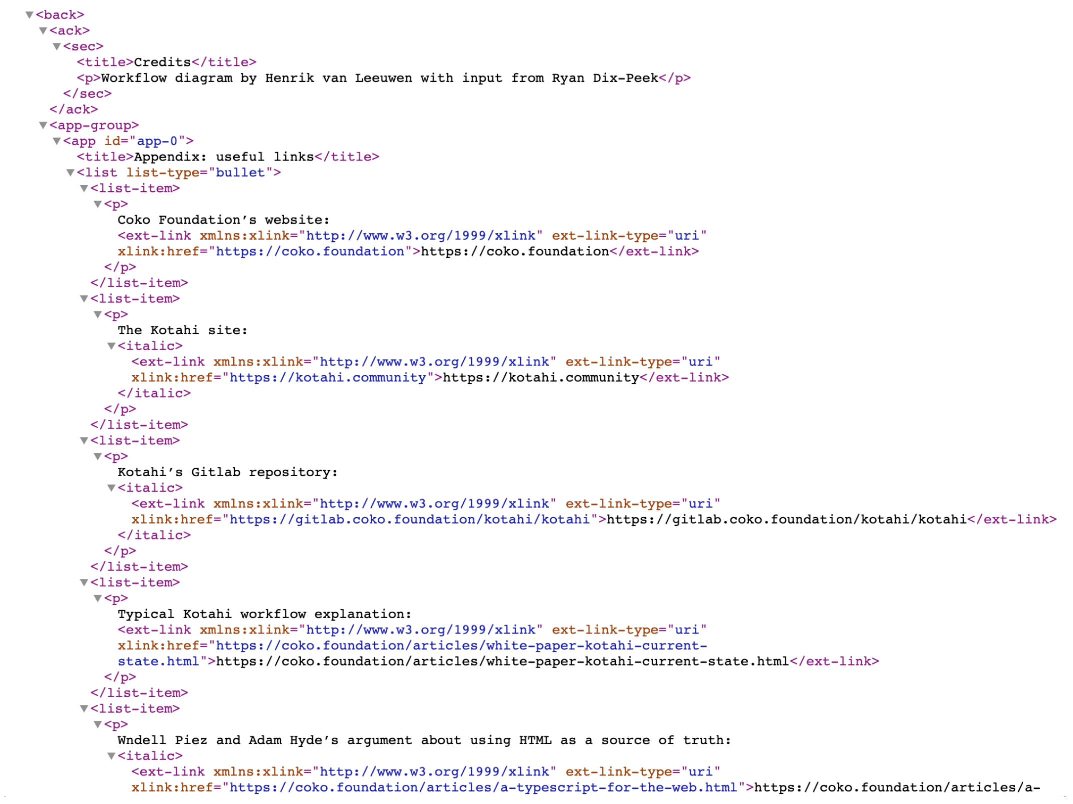
Workflow diagram by Henrik van Leeuwen with input from Ryan Dix-Peek

The following image shows what the underlying HTML source in the Wax-JATS editor looks like for the above example.



```
▼<section class="appendix">
   ::before
   <h1 class="appendixheader">Appendix: useful links</h1>
  ▼<ul>
    ▶<li>…</li>
    ▶<li>…</li>
    ▶<li>…</li>
    ▶<li>…</li>
    ▶<li>…</li>
    ▶<li>…</li>
   </ul>
 </section>
▼<p class="paragraph">
   <br class="ProseMirror-trailingBreak">
 </p>
▼<section class="acknowledgementsSection">
   ::before
   <h2>Credits</h2>
   <p class="paragraph">Workflow diagram by Henrik van Leeuwen with input
   from Ryan Dix-Peek</p>
 </section>
```

Finally, at a click of a button the above HTML is mapped onto a JATS structure. The below image shows the above HTML converted to JATS by Wax-JATS at export time.

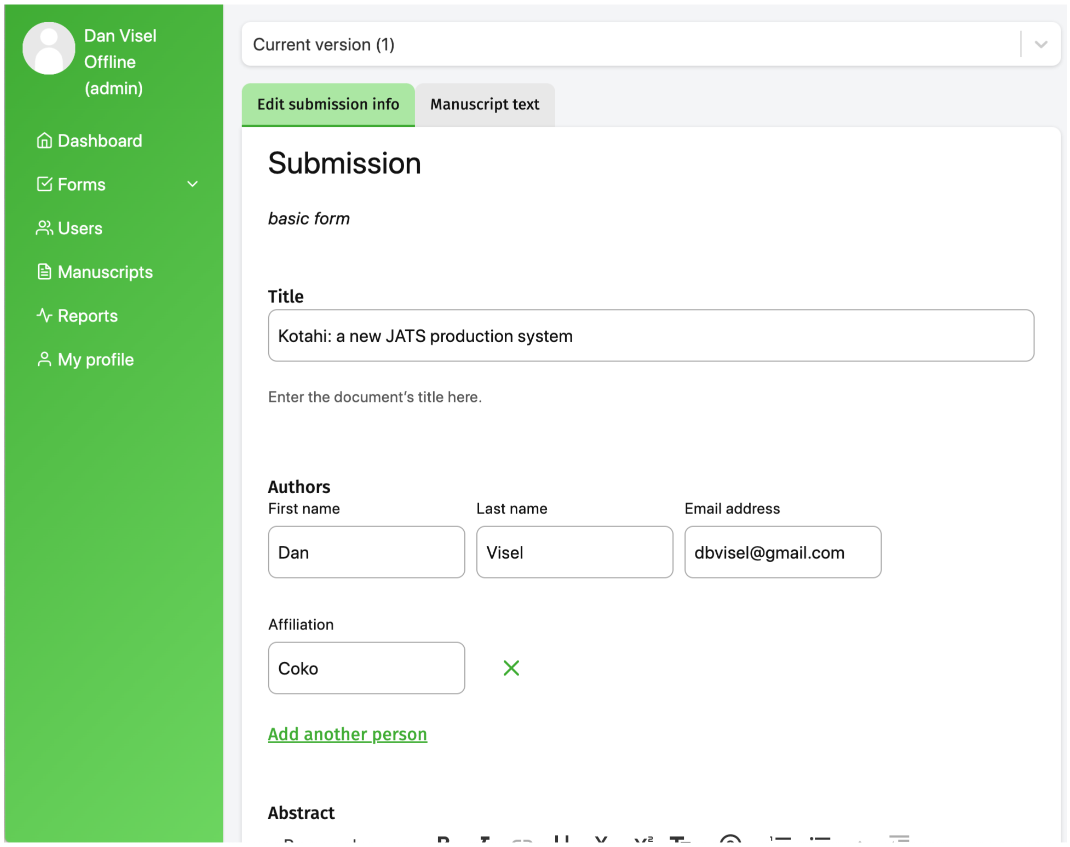


```
▼<back>
  ▼<ack>
    ▼<sec>
       <title>Credits</title>
       <p>Workflow diagram by Henrik van Leeuwen with input from Ryan Dix-Peek</p>
     </sec>
   </ack>
  ▼<app-group>
    ▼<app id="app-0">
       <title>Appendix: useful links</title>
      ▼<list list-type="bullet">
        ▼<list-item>
          ▼<p>
             Coko Foundation’s website:
             <ext-link xmlns:xlink="http://www.w3.org/1999/xlink" ext-link-type="uri"
             xlink:href="https://coko.foundation">https://coko.foundation</ext-link>
           </p>
         </list-item>
        ▼<list-item>
          ▼<p>
             The Kotahi site:
            ▼<italic>
               <ext-link xmlns:xlink="http://www.w3.org/1999/xlink" ext-link-type="uri"
               xlink:href="https://kotahi.community">https://kotahi.community</ext-link>
             </italic>
           </p>
         </list-item>
        ▼<list-item>
          ▼<p>
             Kotahi’s Gitlab repository:
            ▼<italic>
               <ext-link xmlns:xlink="http://www.w3.org/1999/xlink" ext-link-type="uri"
               xlink:href="https://gitlab.coko.foundation/kotahi/kotahi">https://gitlab.coko.foundation/kotahi/kotahi</ext-link>
             </italic>
           </p>
         </list-item>
        ▼<list-item>
          ▼<p>
             Typical Kotahi workflow explanation:
             <ext-link xmlns:xlink="http://www.w3.org/1999/xlink" ext-link-type="uri"
             xlink:href="https://coko.foundation/articles/white-paper-kotahi-current-
             state.html">https://coko.foundation/articles/white-paper-kotahi-current-state.html</ext-link>
           </p>
         </list-item>
        ▼<list-item>
          ▼<p>
             Wndell Piez and Adam Hyde’s argument about using HTML as a source of truth:
            ▼<italic>
               <ext-link xmlns:xlink="http://www.w3.org/1999/xlink" ext-link-type="uri"
               xlink:href="https://coko.foundation/articles/a-typescript-for-the-web.html">https://coko.foundation/articles/a-
```

Even if the user has put a Reference List or an Appendix element in an odd place (e.g. in the middle of a document) this approach means that on conversion to JATS, Kotahi pulls the reference list or appendix out of the body text and moves them to the back matter (conformant with JATS specifications) of the resulting JATS file. This helps prevent the user from making invalid JATS.

JATS structures front matter very carefully in a `<front>` tag. Kotahi can generate this very easily from the internal submission form data, which, for example, will split a list of authors into neatly structured first and last names and affiliations.

The following image is of a basic example Kotahi submission form created by the form builder.



Submission

basic form

Title

Kotahi: a new JATS production system

Enter the document's title here.

Authors

First name	Last name	Email address
Dan	Visel	dbvisel@gmail.com

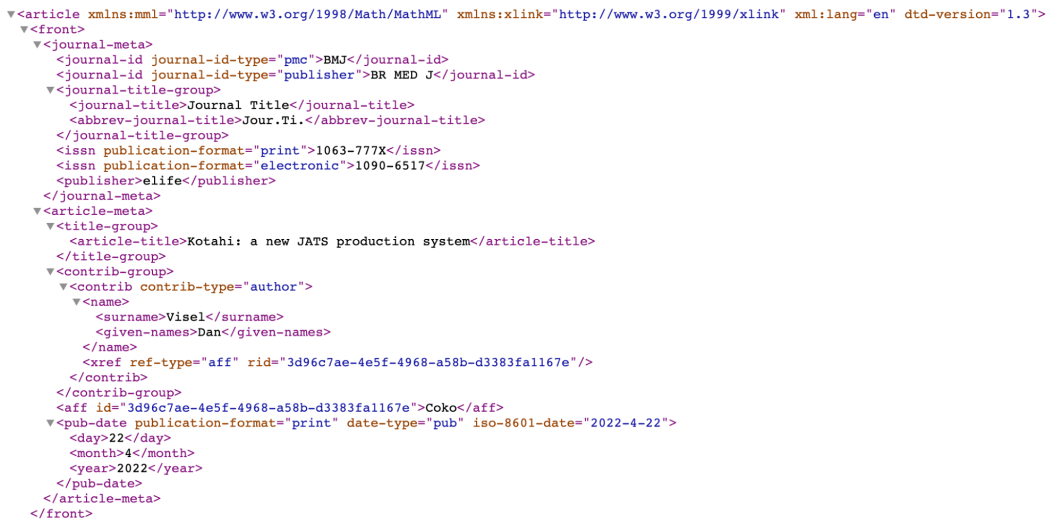
Affiliation

Coko

Add another person

Abstract

The form above displays a few simple fields, all of which are mapped to JATS front matter at export time. The following image shows the data from the above form as it has been exported on-demand to a valid JATS file.



```
▼<article xmlns:mml="http://www.w3.org/1998/Math/MathML" xmlns:xlink="http://www.w3.org/1999/xlink" xml:lang="en" dtd-version="1.3">
  ▼<front>
    ▼<journal-meta>
       <journal-id journal-id-type="pmc">BMJ</journal-id>
       <journal-id journal-id-type="publisher">BR MED J</journal-id>
      ▼<journal-title-group>
         <journal-title>Journal Title</journal-title>
         <abbrev-journal-title>Jour.Ti.</abbrev-journal-title>
       </journal-title-group>
       <issn publication-format="print">1063-777X</issn>
       <issn publication-format="electronic">1090-6517</issn>
       <publisher>elife</publisher>
     </journal-meta>
    ▼<article-meta>
      ▼<title-group>
         <article-title>Kotahi: a new JATS production system</article-title>
       </title-group>
      ▼<contrib-group>
        ▼<contrib contrib-type="author">
          ▼<name>
             <surname>Visel</surname>
             <given-names>Dan</given-names>
           </name>
           <xref ref-type="aff" rid="3d96c7ae-4e5f-4968-a58b-d3383fa1167e"/>
         </contrib>
       </contrib-group>
       <aff id="3d96c7ae-4e5f-4968-a58b-d3383fa1167e">Coko</aff>
      ▼<pub-date publication-format="print" date-type="pub" iso-8601-date="2022-4-22">
         <day>22</day>
         <month>4</month>
         <year>2022</year>
       </pub-date>
     </article-meta>
   </front>
```

Who can produce JATS?

One part of the Kotahi philosophy that should be again noted: we assume that the production team is not necessarily expert in JATS, though the team might have some domain knowledge. Kotahi is aimed at making JATS at scale; a journal might publish thousands of articles per year. While it will take a little time for the production editor to learn how content needs to be formatted in JATS-flavored Wax, the learning curve should not be steep and once the process is familiar the production process is very fast.

Exporting JATS

JATS export is currently done on-demand from the Wax-JATS editor. When JATS export is chosen, the server processes the document's manuscript and submission form data and sends back an XML (JATS) file.

Internally, the 'JATS-flavored' HTML from the Wax-JATS editor is processed tag by tag and turned into JATS markup as covered above.

Initial setup: metadata

The front matter of a typical JATS file composed from an article contains two major types of metadata:

1. metadata specific to the article and
2. metadata for the journal (the journal's title, ISSN, etc.).

Journal metadata (with a few exceptions) tends to be similar across all articles in a journal; this is set up as part of the process of getting an instance of Kotahi off the ground. (Initial setup similarly involves setting the stylistic defaults and layouts for articles as they are seen in the editor and exported as HTML and PDFs.)

The shape of the submission metadata will vary greatly from workflow to work-flow. Kotahi's interactive submission form builder allows setting up the sorts of metadata that will be attached to each submission. However, there's not a one-to-one correspondence between what's in the form and what's exported as metadata. A form may include workflow-specific metadata – who, for example, anonymous reviewers are, something that some users of Kotahi but not others should know – that isn't part of the metadata that's involved in JATS. This is controlled by a mapping of form fields to JATS metadata types during the setup of a new instance of Kotahi (this might change over time towards a friendly admin interface to map metadata to JATS output).

Validation

When JATS is created, it goes through an XML validator and is validated against the JATS schema. Because the elements allowed in Wax-JATS has been carefully enumerated and constrained – all of the paragraph styles, list styles, and character styles that can appear in the document have been defined by configuration - those elements can be thoroughly tested programmatically. Consequently, validation errors are not something that users should see; instead, they tend to indicate that something is wrong on the development end.

JATS as part of the document lifecycle

JATS export (as well as PDF and HTML export) is currently on-demand from Kotahi. Conversion to JATS is not necessarily an end of line process; although it would most likely happen near the end of a document's workflow, a user could generate a JATS file, inspect it, make changes to the manuscript, and regenerate at any time. JATS, PDF, and HTML can all be exported at any time in the document lifecycle.

Crucially, a manuscript that's had JATS elements highlighted can be sent back to the 'normal' Wax editor; so if a last-minute mistake is discovered that needs to be corrected by an author, the author can work in the same editor that review was done in, rather than using a separate (and possibly confusing) production tool.

The Future

Kotahi is under heavy development through a consortium of organizations (Coko, eLife, Amnet, Aperture). Kotahi is also being used with real content by real publishers who are providing valuable feedback about the functionality (and provisions for workflows). While Kotahi is open source, it can be customized to be used in for-profit systems: Amnet Systems, for example, is already using a private instance of Kotahi (Reach OA) with customized functionality to fit into their workflow.

JATS is something that's been added to Kotahi as part of this ongoing process. At this time Kotahi is using a small fraction of the range of functionality that JATS allows; as users call for more, we envision adding more. A handful of examples:

- There's a current bifurcation of documents into a form (the metadata) and the manuscript (the body of the text). Metadata is imported into the body content of the final paper at export time. We would like metadata instead to be linked to the document and be editable in both places. Smart components that are part of Wax/Wax-JATS would allow author names, for example, to appear as they should in a PDF or as HTML but to be correctly parsed when exported to JATS. Editing in the Wax-JATS widget would then also update the form metadata and vice versa.

- Kotahi's treatment of citations is very simple – it's possible to imagine adding a citation manager to Kotahi's interface that would treat citations as structured data if authors can provide them this way. Though there are other ways of achieving this goal: we're also looking at using packages such as ScienceBeam (now a Coko product) to process submissions and automatically extract metadata such as citations to structured data for importing into the form and document.
- Math in Kotahi is currently handled by MathJax, which displays equations as SVGs and exports them as MathML. Other ways of handling math can be imagined and could be added. Interestingly MathJax also supports chemical symbols, which might lead to Wax support of chemistry equation creation using (for example) the mhchem syntax.
- Export process improvements can be imagined. As mentioned above, for example, Kotahi's JATS export is currently on demand; we imagine this will move to a more persistent basis, with current JATS for published documents available via API. Kotahi could also be configured to deliver JATS to predefined third-party services in demand (Kotahi currently does this for delivery of HTML and PDF to the Coko FLAX product and adding JATS to this process will occur in the near future).
- As previously mentioned, the Wax-JATS editor allows users a wide amount of flexibility when identifying JATS structural elements. There's nothing stopping a user, for example, from putting in 12 abstract sections, even though a JATS document can only have a single `<abstract>` tag. Currently if you have a manuscript in the production editor with 12 abstract sections, the first will be pulled out of the body and moved to the front matter of the created JATS and put inside an `<abstract>` tag. The others will be deleted, as abstracts can't appear in the body (according to the JATS specification). The result is perfectly valid JATS, though this result might be confusing to a novice user. The plan going forward is to offer structural hints to the user: So if, for example, a second abstract element is inserted into the text, it might get a red border, and a warning might explain that a document can only have a single abstract in JATS. There is also the possibility to programmatically constrain the number of abstracts identified (etc). However, while Wax does provide the functionality to constrain (for example) the addition of multiple abstracts, this kind of logic is complex. The preference is to take a less burdensome development path in the near term and support the user with information they can use to better structure the document.
- While Kotahi currently uses an on-demand model for creating JATS – where the XML is created and then destroyed after download – it's possible that we'll move to a model where JATS is created automatically

every time a change is made in the production editor; this would happen server-side, and if the editor chose to download JATS, the file would already be ready.

- Until very recently, Kotahi handled uploaded / in a document as base-64 strings. This is in the midst of changing, and this won't be the case within weeks of this article being published. If images are handled as base-64 strings, however, JATS can be exported as a single XML file, which is the current situation. A full-function image store is currently being built; when this is in place, what's output as JATS will not be a single XML file but rather a ZIP file containing the XML and the images in multiple formats – if authors have TIFFs as their original format, for example, the JATS will include both the TIFFs and lower-resolution WEBPs created to display the image online or in PDFs.

Conclusion

JATS is one piece of Kotahi; it's recognized as being part of the process of scholarly publishing and Kotahi integrates it as such. There are certainly other ways of producing JATS than the 'Kotahi way'. The virtue of Kotahi's approach to JATS production – and perhaps why it is valuable to the wider landscape of JATS – is that it makes the threshold to generate JATS very low, and might bring JATS to a wider base of users, and the process is fast, scalable and cost effective.

Kotahi is a modern scholarly publishing platform. It is built with modern technologies with modern, efficient workflows in mind. The platform is also very modular, extensible, and configurable. There is almost no element of the system that is immutable – all elements can be changed if required. In addition, Kotahi leverages modern real-time communication protocols and, together with the Single Source Publishing design, the entire system offers publishers enormous futureproofing and workflow optimization opportunities.

All aspects of Kotahi are open source and liberally licensed (MIT). The Coko Foundation exists to support organizations wanting to hire us to extend Kotahi or go it alone. We also welcome anyone wanting to join the Kotahi consortium/community. In all development scenarios, Coko facilitates the multiple organizations extending Kotahi to ensure the community is building upon each other's work – this helps ensure lowering the burden of development cost and time as well as eliminating possible duplication of efforts.

Additional Useful links

- Coko Foundation's website: https://coko.foundation

- The Kotahi site: https://kotahi.community
- Kotahi's Gitlab repository: https://gitlab.coko.foundation/kotahi/kotahi
- Typical Kotahi workflow explanation: https://coko.foundation/articles/white-paper-kotahi-current-state.html
- Wendell Piez and Adam Hyde's argument about using HTML as a source of truth: https://coko.foundation/articles/a-typescript-for-the-web.html
- Wendell Piez, "Uphill to XML with XSLT, XProc … and HTML," explaining the thinking behind xSweet: https://www.balisage.net/Proceedings/vol20/html/Piez02/BalisageVol20-Piez02.html
- Wendell Piez, "HTML First?: Testing an alternative approach to producing JATS from arbitrary (unconstrained or "wild") .docx (WordML) format," on early attempts to make JATS from HTML with xSweet: https://www.ncbi.nlm.nih.gov/books/NBK425546/
- Adam Hyde, "One Enormous Step at a Time – Now JATS," explaining the different systems behind Kotahi: https://www.adamhyde.net/one-enormous-step-at-a-time-now-jats/
- Adam's Hyde article about single-source publishing: https://coko.foundation/articles/single-source-publishing.html

Credits

- Dan Visel wrote most of this article with edits by Adam Hyde and input from Ben Whitmore. Large parts of the article are quoted from prior articles written by Wendel Piez and Adam Hyde. The section on Single-Source Publishing is a direct copy from an earlier article on the topic by Adam Hyde.
- Diagrams by Henrik van Leeuwen. Initial Aperture and PRC Workflow diagrams by Ryan Dix-Peek (Kotahi Project Manager).
- Christos Kokosias is the lead wax developer for Coko. Wax itself is built on the wonderful open source ProseMirror framework.
- Dan Visel has done most of the work on the Wax-JATS editor (sponsored by Amnet).
- Ben Whitmore is the Coko lead dev for Kotahi. Prior to Ben the lead Kotahi developers for Coko were Giannis Kopanas and Jure Triglav.
- Adam Hyde and Wendell Piez are the designers of the xSweet approach. Wendell Piez did most of the XSLT development for xSweet.
- Adam Hyde designed the Wax-JATSapproach.
- Copy editing by Raewyn Whyte.