How we tried to JATS XML

Obiajulu Odu and Aysa Ekanger

Abstract

This is a story about how an Open Journal Systems-based library publishing service tried (and failed) to implement XML in one of its publications. We ran a small project to look at how journals we support could develop a JATS XML-based publishing workflow using existing open software tools.

Introduction

Septentrio Academic Publishing is a library-run service at UiT The Arctic University of Norway that publishes open access journals on the Open Journal Systems (OJS) platform.[1] Septentrio has always strived to optimize its journals for the digital format, by looking at issues such as screen-friendly PDF layout, long-term archiving, and visibility in the open access infrastructure through journal indexing and permanent identifiers.

If you only publish journal articles in human-readable formats, like PDF, you are likely to be missing valuable indexing opportunities. JATS-compliant XML is increasingly the standard indexing format. Publishing articles in XML format has long been a discussion in the Septentrio team because of XML's benefits and advantages. However, limited resources of the service have prevented the team from bringing the plans to fruition.

When the University Library decided to publish an issue of the Septentrio-series *Ravnetrykk* dedicated to Stein Høydalsvik, the Septentrio team thought: how about publishing this issue in the XML format? Stein coordinated the Septentrio service for a number of years and it would be a nice tribute to Stein if Septentrio's first XML experience was an issue dedicated to him.

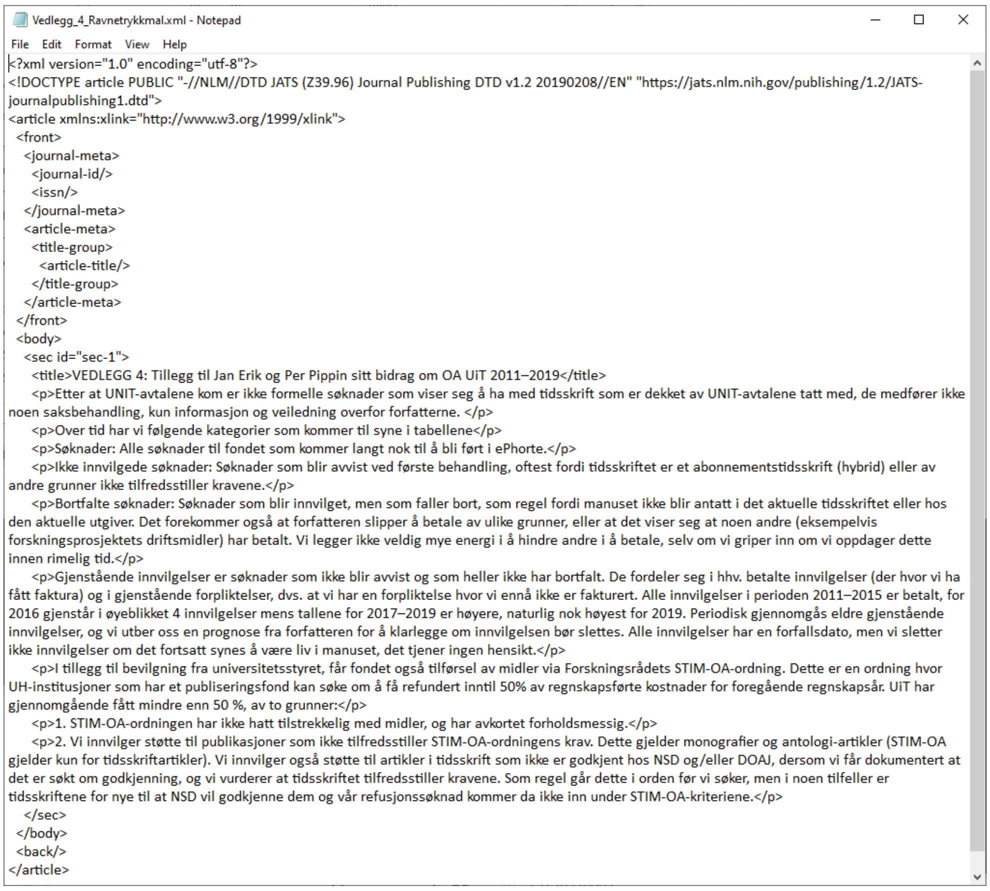
[1] OJS was released by the Public Knowledge Project (PKP) in 2001 and is, according to PKP, "the most widely used open source journal publishing platform in existence" (https://pkp.sfu.ca/ojs/).

https://doi.org/10.7557/15.5517

In this short article we give an overview of the work that was done to publish an issue in *Ravnetrykk* using JATS XML format. We explain the choices made, challenges met and experiences gained.

Reviewing of JATS tools for XML Publishing

Extensible Markup Language (XML) is a markup language that allows to encode content in a machine-readable and layout-independent way, which is more flexible and reusable for a variety of formats (PDF, HTML, etc.). XML makes documents more searchable and accessible, and better-suited for preservation; it allows text mining and opens for content enrichment through multimedia and semantic tagging.[2] The XML format has long been a discussion in the Septentrio team because of its benefits.


Vedlegg_4_Ravnetrykkmal.xml - Notepad
File Edit Format View Help

```xml
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE article PUBLIC "-//NLM//DTD JATS (Z39.96) Journal Publishing DTD v1.2 20190208//EN" "https://jats.nlm.nih.gov/publishing/1.2/JATS-
journalpublishing1.dtd">
<article xmlns:xlink="http://www.w3.org/1999/xlink">
 <front>
  <journal-meta>
   <journal-id/>
   <issn/>
  </journal-meta>
  <article-meta>
   <title-group>
    <article-title/>
   </title-group>
  </article-meta>
 </front>
 <body>
  <sec id="sec-1">
   <title>VEDLEGG 4: Tillegg til Jan Erik og Per Pippin sitt bidrag om OA UiT 2011–2019</title>
   <p>Etter at UNIT-avtalene kom er ikke formelle søknader som viser seg å ha med tidsskrift som er dekket av UNIT-avtalene tatt med, de medfører ikke
noen saksbehandling, kun informasjon og veiledning overfor forfatterne. </p>
   <p>Over tid har vi følgende kategorier som kommer til syne i tabellene</p>
   <p>Søknader: Alle søknader til fondet som kommer langt nok til å bli ført i ePhorte.</p>
   <p>Ikke innvilgede søknader: Søknader som blir avvist ved første behandling, oftest fordi tidsskriftet er et abonnementstidsskrift (hybrid) eller av
andre grunner ikke tilfredsstiller kravene.</p>
   <p>Bortfalte søknader: Søknader som blir innvilget, men som faller bort, som regel fordi manuset ikke blir antatt i det aktuelle tidsskriftet eller hos
den aktuelle utgiver. Det forekommer også at forfatteren slipper å betale av ulike grunner, eller at det viser seg at noen andre (eksempelvis
forskningsprosjektets driftsmidler) har betalt. Vi legger ikke veldig mye energi i å hindre andre i å betale, selv om vi griper inn om vi oppdager dette
innen rimelig tid.</p>
   <p>Gjenstående innvilgelser er søknader som ikke blir avvist og som heller ikke har bortfalt. De fordeler seg i hhv. betalte innvilgelser (der hvor vi ha
fått faktura) og i gjenstående forpliktelser, dvs. at vi har en forpliktelse hvor vi ennå ikke er fakturert. Alle innvilgelser i perioden 2011–2015 er betalt, for
2016 gjenstår i øyeblikket 4 innvilgelser mens tallene for 2017–2019 er høyere, naturlig nok høyest for 2019. Periodisk gjennomgås eldre gjenstående
innvilgelser, og vi utber oss en prognose fra forfatteren for å klarlegge om innvilgelsen bør slettes. Alle innvilgelser har en forfallsdato, men vi sletter
ikke innvilgelser om det fortsatt synes å være liv i manuset, det tjener ingen hensikt.</p>
   <p>I tillegg til bevilgning fra universitetsstyret, får fondet også tilførsel av midler via Forskningsrådets STIM-OA-ordning. Dette er en ordning hvor
UH-institusjoner som har et publiseringsfond kan søke om å få refundert inntil 50% av regnskapsførte kostnader for foregående regnskapsår. UiT har
gjennomgående fått mindre enn 50 %, av to grunner:</p>
   <p>1. STIM-OA-ordningen har ikke hatt tilstrekkelig med midler, og har avkortet forholdsmessig.</p>
   <p>2. Vi innvilger støtte til publikasjoner som ikke tilfredsstiller STIM-OA-ordningens krav. Dette gjelder monografier og antologi-artikler (STIM-OA
gjelder kun for tidsskriftartikler). Vi innvilger også støtte til artikler i tidsskrift som ikke er godkjent hos NSD og/eller DOAJ, dersom vi får dokumentert at
det er søkt om godkjenning, og vi vurderer at tidsskriftet tilfredsstiller kravene. Som regel går dette i orden før vi søker, men i noen tilfeller er
tidsskriftene for nye til at NSD vil godkjenne dem og vår refusjonssøknad kommer da ikke inn under STIM-OA-kriteriene.</p>
  </sec>
 </body>
 <back/>
</article>
```

Figure 1. Example of JATS XML

There are a lot of XML-producing tools – we need tools compatible with OJS, and namely those that produce XML in JATS format. The Journal Article Tag Suite (JATS) is an XML tag set format to

[2] See more at https://en.wikipedia.org/wiki/XML. An overview of the benefits of XML can be found in a blog post by Shanu Kumar, *JATS XML: Everything a Publisher Needs to Know*, https://blog.typeset.io/jats-xml-everything-a-publisher-needs-to-know-95862a4184a3.

structure, share and archive academic articles, that has become an international standard.[3] The tools should also be open source: proprietary solutions were intentionally excluded in our project because of Septentrio's overall commitment to openness and freedom of knowledge. It was also preferable to choose a tool that converts DOCX (Word) files to XML, as all authors in the *Ravnetrykk* issue (and almost all authors of the rest of Septentrio journals) submit their manuscripts as DOCX files.

A good starting point for our project were tools presented at the XML Publishing Workflow workshop that took place at the PKP 2019 International Scholarly Publishing Conference.[4]

The picture below depicts the workflow for JATS XML in Septentrio. We divided the JATS workflow in four stages: submission, conversion, editing and presentation/visualization stages. XML can be introduced at different stages of the production process, with XML-first, XML-last and XML-middle workflows.

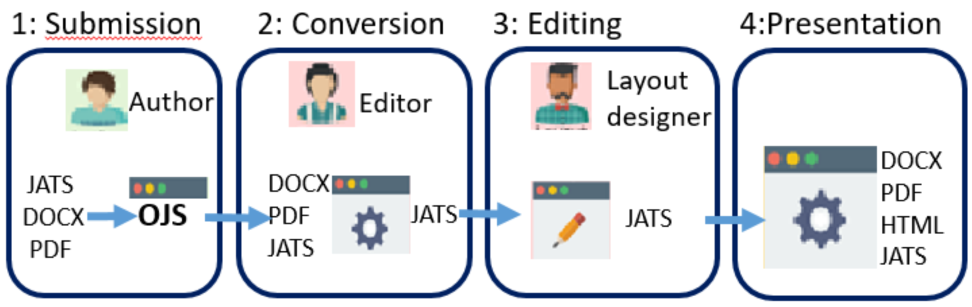


Figure 2. Stages of JATS XML workflow

Making Choices in Different Stages

Our search for a free software tool that is robust, perdurable and standard-compliant, consisted of soliciting help in the PKP Community Forum and testing tools on GitHub compatible with OJS 3.1/3.2 or later. We excluded tools that did not show any sign of development in the last two years or seemed abandoned.

Submission stage

As mentioned above, we decided to focus on XML tools that work with the DOCX format as this format is used by the authors of most of Septentrio's journals and series.

Septentrio also has two journals where the preferred format of submissions is LaTeX; if an XML workflow is to be developed for all of Septentrio's journals in the future, we need to consider tools that can work with LaTeX.

[3] https://en.wikipedia.org/wiki/Journal_Article_Tag_Suite
[4] We briefly discussed the workshop in Eikebrokk et al. (2019), https://doi.org/10.7557/11.5204.

Conversion stage

The originals (DOCX-files) from authors are transformed to JATS XML. We did a review of two tools that work with DOCX files: meTypeset and docxToJats.

meTypeset[5]: is a standalone command-line tool written in Python to convert from DOCX format to JATS XML. It requires the text to be formatted in a particular way and gives mixed results. Last changes in the development seemed to have taken place more than a year before we started reviewing the tool.

docxToJats[6]: The tool works as a standalone application written in PHP that allows making batch transformations of articles from DOCX to JATS XML. The tool, which is being actively developed, is still a «work in progress», so a lot of manual cleaning of the output XML file is needed. docxToJats and its OJS plugin, JATS Parser, seemed the most appropriate for our use case as it has the most recent developments.

Editing stage

The docxToJats tool is a «work in progress», not all features usually present in a manuscript (e.g. figures, formulas and footnotes) are supported. If a conversion is not perfect, or some modification must be made in the final galley, a tool is necessary to make changes directly in the produced JATS XML files.

For this stage of the workflow, a relevant tool is Texture[7] – a visual JATS XML editor that allows authors/editors to produce documents in JATS from scratch or edit existing files. It has a simple, user-friendly graphical interface and it is available as a standalone tool and as an OJS plugin. Note, however, that it supports only a strict subset of JATS elements, which some may find too restrictive. A thorough test of the tool for this purpose is pending.

Presentation stage

JATS XML needs to be converted to something more human-readable, like HTML or PDF. We tested two OJS3 plugins: Lens Galley viewer plugin and JATS Parser plugin.

Lens Galley viewer plugin[8] integrates eLife Lens for OJS 3.0. Lens Galley viewer is a well-developed tool that displays the full text in a separate window, with flexible navigation options. However, it is not optimized for display on mobile phones and tablets. It did not work

[5] https://github.com/MartinPaulEve/meTypeset
[6] https://github.com/Vitaliy-1/docxToJats
[7] https://github.com/pkp/texture
[8] https://github.com/asmecher/lensGalley)

in OJS 3.1 and 3.2, and it collides with Texture plugin (as pointed out in the PKP Community Forum).

The JATS Parser plugin[9] parses JATS XML and displays it on the article's abstract page in OJS (the landing page for the article's DOI) as HTML. The plugin also allows to opt for auto-generation of a PDF from the XML: a PDF link is then shown on the article abstract page. The plugin parses JATS content (<body> and <back> sections), whereas metadata, such as title, author and article abstract, are extracted from the submission metadata in OJS. Elements such as figures, formulas and footnotes still need to be implemented in the plugin. JATS Parser does not work in all OJS themes, and works on OJS 3.2.x, but not on OJS 3.1.x (which Septentrio currently is based on). Nevertheless, despite these shortcomings, JATS Parser at the moment is the best solution for the Presentation stage.

Challenges and Experiences

To-do lists at our library department are usually long, and the spring of 2020, with its COVID-19 disruptions, did not contribute to making them shorter or easier to move through – rather the opposite. The editor-moderated final versions of the manuscripts started arriving by the beginning of May 2020 and it was about that time that the exploration of XML tools started. The team have not achieved fully the implementation in the mentioned four stages due to shortage of time and the limited resources of the service. Nevertheless, this was a useful learning experience for us, and potentially useful for others who intend to implement the workflow.

docxToJats did not support all major features of DOCX.[10] The planned 1.0.0 release will likely be the first stable release.

When a DOCX file is processed, the text is converted into an XML file, whereas non-textual elements like images, embedded Excel objects, tables and figures are extracted into attached files. These attached files must be uploaded as dependent files to XML galleys in OJS.

[9] https://github.com/Vitaliy-1/JATSParserPlugin

[10] An overview of the elements that are already supported and are planned to be developed in the near future is available at https://github.com/Vitaliy-1/docxConverter#what-article-elements-are-supported.

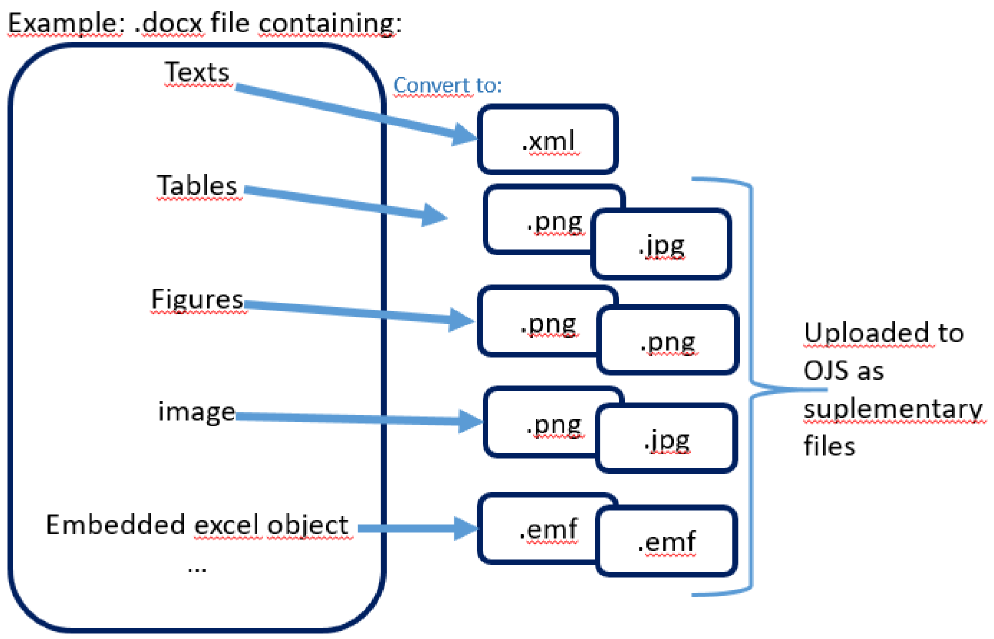


Figure 3. From a DOCX file to an output XML file and attached files

Each dependent file in OJS has associated fields for captions and rights information. Table and figure captions functionality in the docxToJats plugin is planned for the 1.0.0 release; it will be interesting to see whether there will be an option in the JATS Parser plugin to choose where the captions information must be imported from – the XML file or dependent file metadata.

Septentrio is currently running on OJS 3.1.2.0. We started testing XML conversion in a test environment running 3.2.0.2, which happened to be a newer version of OJS. The initial results were encouraging: the articles (consisting of text and images) were converting and displaying nicely. There were a few issues with the non-textual elements not supported by the plugin: but we could find temporary workarounds. We decided to continue in the Septentrio production server and work out the possible remaining issues there. To our disappointment, however, the JATS Parser plugin did not work in OJS 3.1.2.0, and we did not find any evidence that the plugin works or has been tested on OJS 3.1.2.0. It was now a little over one week before the issue was supposed to be published. If we wanted to present Stein with an issue in the XML format, we had to upgrade our entire OJS installation, with all the 19 journals and series. An update at such short notice, however, would probably be stressful for the editors of other Septentrio journals, who would have to put up with at least a day of downtime they had not planned for. Another complicating factor was that we lacked a complete Norwegian translation for the new OJS version. This meant that even if the Septentrio team put in extra work hours to take care of the translation files in the following week, authors, editors and reviewers of Septentrio journals would be met with unfamiliar features and possible upgrade hiccups.

In Norway they say: “Det er ingen skam å snu” – there is no shame in turning back. So we terminated this trip and turned back to the PDF-only format. Next time we are better prepared.

Lessons learned

Despite the disappointing failure, we now have a good understanding of open source options for XML-based publishing and the relative strengths and limitations of the different available solutions. We have developed a workflow that, although imperfect and still requiring some manual corrections, could be used to develop an implementation tailor-made for editors who want to incorporate XML into their production processes. A more thorough and extensive future project could include the following:

- Find out if any other institutions have undertaken similar projects and are willing to share their experiences and insights with us.
- Investigate and compile a list of all OJS-compatible open source tools for converting Septentrio scholarly documents to XML.
- Review, test and evaluate the available tools on latest version of OJS before upgrading.
- Edit XML files and render XML into PDF and HTML.
- Validate XML files resulting from different tools.
- Analyze existing documents in Septentrio (published journal articles) to identify potential problems.
- Test tools with DOCX, Google Doc, OpenOffice, and LibreOffice sample files. LaTeX files should also be tested.
- Try to combine the tools into a workflow and take note of points that need attention.
- Identify areas of improvement for the tools selected.
- Produce an initial guide and template for authors and editors.