Now that we have our high-level musings on accessible publishing in libraries out of the way (check out [the ‘accessibility’ category](https://librarypublishing.wordpress.com/category/accessibility/) to see the previous posts on the topic), I think it’s time to start talking about the nitty-gritty. Sure, we all agree that providing accessible digital content is *good,*but how do we *do*it? It should be pretty obvious by now that I’m not an expert on the subject, but I’ve learned a few things that I’d like to share. I’m starting off with what I know best – publishing in HTML – but I’m hoping to also write about accessible publishing in PDF form, as well as accessibility in retrospective journal digitization projects.

**Some background**

Much of my experience with making web content accessible comes from working with [*Disability Studies Quarterly*](http://dsq-sds.org/index) (*DSQ*).  DSQ was the first journal to partner with [OSU Libraries’ Publishing Program](https://library.osu.edu/projects-initiatives/knowledge-bank/open-access-publishing/), and as a result, it has been strongly influential in how our program has developed. Because of the field of study, accessibility was front-and-center from day one: the journal content had to be accessible to readers, and the journal platform had to be accessible to authors, reviewers, and editors. Unfortunately, I wasn’t around for the initial work with Open Journal Systems, so I don’t know what the conversations about platform accessibility looked like. If we were to adopt a new publishing platform today, I would ask the director of [OSU’s Web Accessibility Center](http://ods.osu.edu/faculty-staff/web-accessibility-center-wac/) (in the Disability Services Office of Student Life) to check it out, so maybe that’s what happened.

**Choosing a format**

What I do know is that, to avoid the accessibility problems you tend to find in PDFs (more about that in a future post), we decided to take on the labor-intensive task of converting *DSQ* articles into HTML for publication. Plain, well-formed HTML has a number of advantages over other formats, including built-in structure (important for screen readers) and the ease with which text can be resized. It is also platform-neutral, non-proprietary, and more or less human-readable. Now, using HTML is not a panacea for accessibility – if it were, we wouldn’t have [massive guidelines for how to create accessible HTML](http://www.w3.org/TR/WCAG20/). HTML is also not the only format to offer many of these benefits, but it was something that we felt we could do, and do accessibly.

**Creating HTML**

Did I mention that HTML layout editing is labor-intensive? Here is the process we use:

* **Make some changes in Word:**Starting with a copyedited article in Microsoft Word, I run a series of Word macros to make some basic changes, such as converting ampersands, em dashes, accents, and other special characters into HTML entities; adding <em> tags around italicized text, and converting foot- and endnotes into HTML. This step saves a lot of time in manual HTML keying. We use macros because the person who set up the process was really into them, but it could be done using a variety of scripting methods.
* **Add the content to a template:**Starting from a template (we have one for peer-reviewed articles, one for reviews, one for the editor’s introduction, etc.), I paste in the content from the Word document. Title, author info, keywords, and abstract go into structured fields, and then the rest of the content is housed in nested <div> tags for sections and subsections. When your macro to put <p> tags around all paragraphs in the document is working (ahem), this is a pretty quick process, for the most part.
* **Code lists, tables, and other structured text:** A lot of this has to be done by hand, unfortunately. I’ve found a variety of workarounds over the years, but currently none of them work any better than just hand-coding. For a while there, it was possible to paste most of this stuff into our Confluence wiki software’s WYSIWYG interface and pull it out as beautiful, clean HTML. That was nice while it lasted.
* **Handle images:** If you want people who use screenreaders to access image-based content in the journal, it’s important to include text descriptions. The ‘alt’ attribute of the HTML <img> tag is useful for this, but it’s best to keep it short (~125 characters). If you need a longer description (for, say, a complicated diagram), you can also include a link to an extended text description at the end of the article. Check out [this DSQ article](http://dsq-sds.org/article/view/4938/4027) for an example of what it looks like. (More info about this and about how to use the alt attribute can be found on [this accessibility site from Penn State](http://accessibility.psu.edu/images/imageshtml/).)
* **Check the work:** I like to open the resulting HTML file in my browser and just skim the whole thing, looking for problems. It can be helpful to have someone else do this if you’ve been looking at the document for quite a while.

This isn’t the only way to handle this process. One of the editors for another journal we work with ([International Journal of Screendance](http://screendancejournal.org/)) codes each essay in [Markdown](https://daringfireball.net/projects/markdown/), and then uses [Pandoc](http://pandoc.org/)to convert the Markdown into HTML and Word. The important thing isn’t how you do it – just find a set of tools that works for you.

**Challenges**

The fundamental challenge with converting Word documents to HTML in the context of journal publishing is dealing with the idiosyncracies of individual authors’ use of word processing software. Microsoft Word provides plenty of tools for creating structured documents and well-formed tables, lists, etc. Most people, however, don’t use them much, and authors of scholarly articles are no exception. In the years I’ve been doing HTML layout editing, I have seen a wide variety of creative uses of Word, many of which require extra editing time to convert into something useful. Some of the most common:

* **Headings:**Often, parsing out the section/subsection/sub-subsection structure of a manuscript is an exercise in judgment – piecing together clues (font sizes, italics, bolding, text alignment) to determine what the author intended. I don’t remember the last time I came across an article whose author made use of Word’s built-in header structure.
* **Data:**  I come across everything from nicely-constructed tables to messy clumps of data wrangled into table-like form by repeated uses of the space bar and carriage return. Sometimes data that could be presented in a simple table is included as an image, and sometimes data in a table is organized through text alignment, etc.
* **Notes:** Authors are usually pretty good about using the foot- and endnote functionality in their word processer, but occasionally I come across a paper with hand-generated notes.

Again, none of these problems are unique to HTML. You would run into the same issues while converting Word files to accessible PDFs – it’s just the solutions that differ. I like being able to start from scratch with clean HTML code when the original is too big a rat’s nest, and it’s amazing how many problems you can solve with some extremely rudimentary HTML knowledge.

**Automation on the horizon?**

PKP has been working for a couple of years now on an automated [XML Parsing Service](http://pkp-udev.lib.sfu.ca/), which will convert Word documents into XML, which can then be used to output HTML and PDF formats. You can read about the development and functionality of this service in more detail in a JATS-Con proceedings paper by Alex Garnett, Juan Pablo Alperin, and John Willinsky: [The Public Knowledge Project XML Publishing Service and meTypeset: Don’t call it “Yet Another Word-to-JATS Conversion Kit”](http://www.ncbi.nlm.nih.gov/books/NBK279666/). It’s been over a year since I played around with this still-developing service, so I think it’s probably time for another look. Between a sophisticated approach to converting idiosyncratic Word documents into standard XML and integration with OJS’s workflows, this service provides some hope for OJS users who don’t have the time or staff to do manual HTML conversion, and relief for those of us who are currently working with time-consuming layout editing processes.