A Typescript for the Web

Why HTML is a better starting point than .docx

Published: Dec 5, 2016

Author(s): Adam Hyde , Wendel Piez

Illustrations: Henrik van Leeuwen

PDF Design: Harshna Haswani

Rendered from HTML using Paged.js

Originally written in 2016, small updates in 2022.

In the days of the typewriter, a typescript was a typed copy of a work. The typescript copy was used for improving the document through the editorial process – for reviewing, commenting, fact and rights checking, revision etc.

For many years, since the advent of desktop publishing, the hand-typed typescript has been replaced by Microsoft Word documents. This advanced the publishing world quite a bit. Emailing typescript in the form of Word documents was far easier than mailing paper copies, and making revised copies moved from being an arduous typing exercise to the one click ‘save as’. There are obvious efficiencies.

But now we are in the age of networked documents. We have the opportunity to make another paradigmatic workflow change – in a sense: bringing typescript to the browser. With this evolution, we can bring an end to many of the frustrations of emailing Word documents around for comment and revision – while exploring wider options for collaboration, innovating more interesting content types, more easily understanding and managing a document’s history, using automated typesetting and data exchange. All of these represent cost and time savings for the publishing process and have the potential to move the communication of research beyond the limited paradigm of the manuscript.

However, migrating typescript formats away from the desktop to the web has proven to be very difficult. One of the major problems is that while there is a growing move towards online authoring environments, many authors still start in Word, and many previous attempts have shown that MS Word is not an easy file format to convert to other formats. Fortunately, we believe, *if we think of MS Word as a software for “typescript preparation,”* that HTML is a viable option for conversion from Word’s .docx format into a typescript format ready for the web.

HTML works out well as a format for these purposes due to features that are more commonly considered weaknesses. How so? Well, first we must consider that, at early stages of a publishing workflow, a Word document will not have achieved its final structure, or indeed, much of any structure at all. Nonetheless, attempts to ‘get out of Word’ have tried to jump from unstructured 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. It is a little like going, as Peter Murray Rust has previously commented, from

Hamburger to Cow.

However, with the XSweet Word-to-HTML scripts we are developing, we retain step one (copying over all the data) and replace step two (interpolating structure) with a process that forsakes the introduction of any structure in favor of carrying over whatever information from the original 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 Word document into an unstructured (or partially better structured) HTML document; and
2. interpret the original 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. In an HTML editor, we can then bring to bear further tools for adding structure, reviewing, commenting, revising. Hence, there is a kind of similarity to its historical predecessor – which is why we are using the working title 'HTML typescript'.

If the decision is to improve the structure programmatically, the Coko XSweet platform comes in nicely. A developer can use an XSweet "docx extraction" to get to HTML typescript, and then write (or reuse) another step, in the language of their choice, to do structural interpolation. By separating concerns here, XSweet allows developers to provide structural interpolations that suit the kinds of content their organisations are working with, in the way that makes most sense for them.

When manual improvement is required, we can use sophisticated editors like those built with the ProseMirror libraries (such as Wax). These editor manages strict control of the source and ensure that the manually applied structure is semantically robust. This also means that we can apply explicit structure to the text and co-relate that to the other output formats we require, such as JATS XML as Coko has done with the Wax-JATS editor.

It is not a revolutionary new approach, rather a shift in emphasis. By avoiding over-complicating the conversion as described above, we have been able to make faster headway – and to turn our focus to the tools required to support the editorial processes via a web-based typescript.

Post by Wendell Piez and Adam Hyde.