TWYSIWYGHTMLS!

Evan Brass - 932990994 Quinton Osborn - 932356954

January 15, 2018

Contents

1 What experience could be better?

Documents, particularly technical ones are starting to look more like software. A set of source files is then compiled into a final document. One system that does this is LaTex. HTML and LaTex share two main benefits:

1. LaTex and HTML are both text based which lets them be managed using the powerful text based version control systems.

2. They are open source or open standard

Many of the tools that programmers have developed for their own work solves problems in other fields. Stack Overflow, GIT, and the myriad of testing, automation, and code quality control tools could be used in writing if the format for the document looks like code (or in HTMLs and LaTexs case, is code).

2 Why is typesetting not enough?

Documents can be consumed in a variety of different ways. Accessibility devices, mobile devices (with a widening variety of shapes like circular watches, rounded bezels on phones, and the different aspect ratios of portrait and landscape devices), and search engines all need access to the same content. HTML is designed to transport information of a variety of types and leave its precise display details to CSS or the user agent. This separation of concerns is the best way to deal with the variability between how readers will access your document. HTMLs benefits include:

1. HTML is practically universal. Almost anyone can view HTML documents.

2. HTML and other web technologies have driven users (and readers) expectations for how a document should behave. Anchors both between documents and between applications let the information you see turn into relevant actions.

3. HTML has cross platform accessibility.

4. HTMLs metadata makes it ideal for searching and indexing. HTML tags make what a writer is saying parsable to a computer or accessibility device.

5. HTML can serve as its own editor using contenteditable.

3 So where does LaTex have HTML beat? Where are they different things?

HTML is like the output of a LaTex document. HTML has no functions or any other method of generating content on the fly. Thats one of the problems I want to solve and we can solve it by thinking of HTML as both the .tex file and the final output file in one. Because HTML has so much metadata, I believe that many common document parts (bibliographies, table of contents) can be derived from the HTML document, formatted into insertable HTML parts and then pasted back in at the proper location. In the cases where a document part cant be inferred directly from the HTML, custom properties or the microdata specification could be used to link/annotate the document with enough information to create them.

My hope is that this functionality could be wrapped in an HTML, custom element, editor so that it can reach as many authors as their documents reach readers. Browsers even take care of some of this WYSIWYG editor functionality using the contenteditable attribute. Contenteditable has a long way to go though. Most of the built-in HTML elements have no method of changing all of their powerful attributes. For example, an image tag has no way of resizing itself, input fields have no method of changing their placeholder text, and links have no method of editing their href. This is the first issue that would need to be addressed.

The second would be to create a set of rules or algorithms to define a preference for semantically equivalent HTML. One of the inhibiting factors for the adoption of HTML as a document format is that different editors can use a variety of different markups to describe the same content. Some applications use a single generic tag (say div) and use CSS classes, aria attributes, and javascript to recreate functionality that HTML possesses natively. HTML is very flexible and bends to fit this paradigm at the cost of speed, accessibility (without compensating design), and parsability. This old paradigm will be unnecessary with Shadow DOM (assuming that javascript is available to you) because those sections can be concealed.

Third, after the DOM is flat and parsable, we can create algorithms for things like creating a bibliography and table of contents. This is the

programmatic portion that would make this more like LaTex. Although I doubt it would be necessary, custom attributes could be used to link citations to their bibliographic entries or any other instance where more information is needed.

4 Aren't there enough HTML editors already?

Yes, there are many different WYSIWYG editors, however, I dont know of any that are based on using custom elements to provide editing abilities for built in elements. This method would also lend itself well to editing documents that contain other custom elements.

One of the difficulties when choosing an HTML editor is extending it with site specific features. One option is to include a shortcode like Wordpress does, however this abstraction cant contain all the information that might be required to build a custom element. HTML is the best thing to represent itself.

5 How would the end product function?

My current idea would be to have a top level custom element that creates elements, reduces the html inside to the most terse but semantically equivalent formatting, and handles common editing features like keyboard shortcuts.

Then, the end user can include as many editable elements as they want. (An editable element would be a custom element that provides the interface to edit an element). These elements would register themselves with the top level editor element so that its interface can provide a method to create that element (probably a button in the interface). Each individual element can provide the most correct and meaningful HTML output for itself. Most, if not all of the built-in elements would need custom elements to make their entirety contenteditable. The interface by which they do that (perhaps a little popup) would be hidden using the Shadow DOM. Then, when the time comes to save or publish the document, all of these custom elements (or at least the ones used to edit native elements) can be downgraded (essentially remove most of the is attributes) to their native forms. What would be left is an HTML document with all of the metadata it needs to be re-edited placed in the locations recommended by web standards. To continue harping the

citation example, an editable citation element could automatically populate html attributes to conform to a citation vocabulary like that of Schema.org.