LaTeX is Dead (long live LaTeX)
Typesetting in the Digital Age

Deyan Ginev

Jacobs University Bremen

Welcome to the Pitchfork Party

This post comes in the context of a series of healthy discussion pieces on authoring scientific content for the web:[1]

– LaTeX was not built for the Web by Alberto and Nate from Authorea.
– LaTeX Something Something Darkside by Peter Krautzberger from MathJaX.
– A Scholarly MarkDown discussion on Hacker News (see the comments)

In this text, I will try to elaborate on the merits and deficiencies of using a pre-web authoring syntax, LaTeX, for writing modern publications in 2015 as active web documents. My stance is evolutionary – we should adapt our existing tools to the new environment and in the process gain insights for what the next generation of tools ought to be.

If you are a working scientist who authors in LaTeX, I will suggest how to gradually adapt your existing toolchain, while making your first steps towards the future of publishing. You can skip to my suggestion if you don't find the technical details interesting.

If you are a developer, I will argue with you that the next generation has not fully arrived yet.

We're not going to start a fire

Feelings can burn strong when the words "LaTeX" and "Web" appear together.

Debates over tool superiority, especially when online, tend to quickly become heated and destructive. My best guess is that the personal experiences with our tools over time evolve into full-blown relationships, with all associated pros and cons of that status. Maybe you truly love your tool, and that is great, please go ahead and nourish that feeling. Meanwhile, I will step back into more abstract territory and try to poke some apps with a stick and see when they bite. You're welcome to tag along, but there's no need for extra venom.

Bootstrapping the Basics

LaTeX

LaTeX is an authoring format for documents, built on top of Donald Knuth's typesetting system TeX. It has originally been designed for creating beautifully typeset manuscripts for print, in the "pixels on paper" paradigms of PDF and DVI. Its killer feature? You can literally "program" your text – both its form and function.

Being able to "program" a text is not immediately a good usability feature. In fact, if you are writing in text-heavy genres, such as fiction, or mainly author short forms, such as emails, blogs and Facebook posts, you quite likely don't need anything beyond the most basic word processor.[2] In the similarly vehement debates over the value of programming languages, peace can sometimes be kept via a simple rule of thumb:

[1] The author has his own torch in hand: I am a core contributor to LaTeXML and an enthusiastic developer at Authorea. So keep that bias in mind while reading on.

[2] If you are mostly using Twitter, even a basic tablet on-screen keyboard feels like a waste.

Use the Best tool for the job.

LaTeX can be a "best tool" and has been a very loved partner in writing scientific publications for the Formal and Natural sciences. It has been particularly dominant in math-heavy fields, as beautiful mathematical formulas were a TeX-only prerogative until the early 2000s. Many would argue that the competing Office solutions have been playing catch-up for a long time also for citations and advanced typographic styling. If you were writing a book or technical manual that was well into the hundreds of pages, LaTeX could robustly process and typeset your document from its early inception.

However, great power brings great opportunity for blunders. The cost of LaTeX is a tedious syntax (by modern standards) and a large range of possible errors, caused by the hacked-together module system of classes and styles, and the absence of reliable encapsulation or inheritance. As programming languages go, TeX is one of the hardest to read or write. Unsurprisingly, that makes alternatives quite appealing, especially for uses that do not require a typesetting bazooka.

MarkDown

MarkDown is among the most used authoring languages for web documents, powering Wikipedia and countless specialized wiki-based sites. It is common to use MarkDown as a contrast to LaTeX in order to demonstrate the difference in typesetting paradigms, but also the "Convention over Configuration" approach applied to authoring tools.

Looking back on the inception of projects, we see that MarkDown has targeted web pages (HTML), while LaTeX and Office have targeted printers (DVI and later PDF). But the differences do not stop there, as MarkDown is also much more limited in scope and expressivity, and if you ask too much of it, it pecks back.

It is also common to use one part of the technology stack to refer to the entire toolchain[3]. Namely, MarkDown usually implies creating an HTML document, while LaTeX implies a PDF document. It is important to keep in mind that while they are traditionally used together, these are separate and very different formats that can be re-appropriated. It is possible to both create PDF documents from MarkDown, as well as HTML documents from LaTeX, but those are the current exceptions, rather than the rule.

I will only go into the typesetting aspects in this post, as I plan to write a separate entry for my thoughts on authorship UX. For now it is enough to note that MarkDown is incredibly simple to learn and use, as it has a minimal set of typesetting commands (e.g. sectioning, math, tables). LaTeX on the other hand is a full-blown programming language with a vast ecosystem of extra features and extensions, which allows you to customize Anything.

Finding the sweet spot that strikes the perfect balance between power and convention is also one of our goals at Authorea, where we strive to offer a fluent editing experience to researchers.

Web-first Publications

Roll the tape forward to 2015. Books are still printed and perused, but a great amount of our writing has moved to a web-first form. The language of the web in 2015 is HTML5, and a "web-first scientific document" is an HTML5 document.

While most established publishers still work on a print-first basis, the majority of publications are now available online, at a minimum as legacy PDF documents. From what I can tell, the movement to web-first publishing is growing ever stronger, with the advent of e-Readers and active documents. The promise of active documents[KCD+11], which can embed not only hyperlinks, but also data, is quite appealing. On-demand machine support can offer a wide range of services, from screen-readers and machine translation to interactively exposing the data behind figures and tables.

[3] An example of a technological use of metonymy

The web-first scientific manuscripts of 2015 are HTML5 documents. LaTeX is one of several viable, yet imperfect, authoring languages for the web.

And if you are keeping up, you would correctly notice this implies a significant paradigm shift.

A Gourmet List of Features

Let us start with a handpicked list of high profile typesetting features[4] that we need in either web or print and check the overlap. I will limit myself to 10, for the sake of brevity:

Feature	Web	Print	Example
Citations	✓	✓	`\cite{KohDavGin:psewads11}` = [KCD$^+$11]
Math	✓	✓	$P(E) = \binom{n}{k} p^k (1-p)^{n-k}$
Labels/Refs	✓	✓	`See Fig.\ref{fig:xkcd}` = See Fig. 1
Metadata	✓	✓	Author, Title, License, ...
Figures	✓	✓	
Tables	✓	✓	$\begin{array}{\|c\|c\|} \text{a} & \text{b} \\ \text{c} & \text{d} \end{array}$
Fonts and Styles	✓	✓	
Page/line-breaks	×	✓	
Semantics	✓	×	Screen-readers for math
Embedded Data	✓	×	iPython notebooks

Table 1. A Gourmet Selection of Typesetting Features

It is probably safe to conclude that print and web documents have large overlap in both content and function. This statement feels intuitive to me, as the documents convey the same message, written for the same readers. But while the end goals are similar, the path to reaching them is quite abruptly different.

Turning the Page into a Playground

Pixels and Trees

As Alberto and Nate point out in LaTeX was not built for the Web, all LaTeX commands eventually expand into TeX instructions that target a printed page, which is just an array of pixels. Behind the curtain, TeX undergoes an elaborate process of forming and stacking horizontal and vertical boxes, computing line- and page-breaks and carefully aligning and kerning various structures. The final result is a list of points with absolute positioning, in a printer-friendly language.

The design of a web document is fundamentally based on a **paradigm shift**, away from absolute positioning. Web pages are intended to be dynamic and fluent – they are born resizeable and customizable to changes in fonts and styling in general. HTML5 separates content, form and behavior by allowing for modular interplay between HTML, CSS and JavaScript. This view of a document is distinctively more abstract than the printer-friendly one, and focuses on a document's **structure** rather than the low-level details of its display.[5]

[4] Typesetting concerns itself with the final **appearance** of the document, rather than the authoring process

[5] As it turns out, the narrative structure of a document is best represented by trees, which are one of the core datastructures in Computer Science that we all love and cherish. Web developers who have come to love traversing a DOM can appreciate that fully.

Bruce Miller, creator of LaTeXML, refers to this paradigm tension as an "**impedance mismatch**". It is not always possible to directly align our HTML expectations of block, table, inline-block and inline structures with the LaTeX reality of absolutely positioned boxes, filled with virtual ink. Tools such as LaTeXML and Pandoc aim to capture, and at times guess, the underlying structural semantics of a LaTeX manuscript. When they succeed, we can successfully create beautiful, high quality web documents at no extra cost to the author, in addition to the usual PDF result. In the many cases where the tension is too great, there are two ways out - either forbid the author the use of print-only commands, or dedicate developer time to provide special support for those commands.

On the other hand, how can one obtain printable versions of MarkDown documents? That is usually achieved either through using a web browser's native printing capabilities on top of the final HTML document, or via a vanilla style conversion by dedicated tools, such as Pandoc.[6]

Pointing Makes Perfect

A crucial aspect for both classic and web documents has been the ability to label and reference various bits and pieces of interest. Being able to easily and consistently name Table 1 and Figure 1 is basic hygiene when writing a book, as we keep adding and reordering these pieces as the writing process goes on. In academic texts, it is also of crucial importance to be able to cite passages of relevant literature, without getting completely lost in the process – many texts have tens and some even hundreds of citations. In web documents we also have a second type of external references that we are well familiar with – hyperlinks.

This is one area where LaTeX has traditionally performed exceptionally well and MarkDown has inherited a flavor of its capabilities, reaffirming their value.

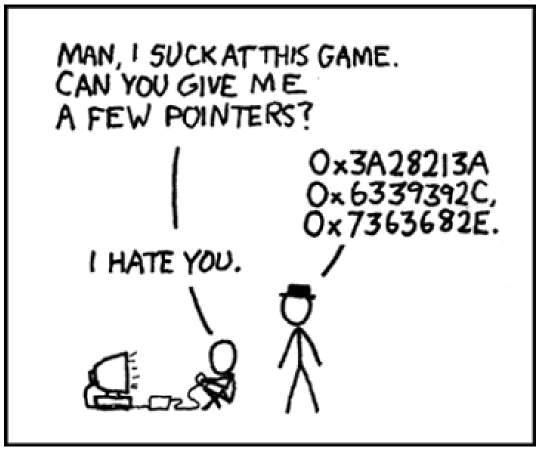
MAN, I SUCK AT THIS GAME. CAN YOU GIVE ME A FEW POINTERS?
0x3A28213A 0x6339392C, 0x7363682E.
I HATE YOU.

Fig. 1. No, not this kind of pointers. [Source: XKCD]

I want to illustrate one aspect of progress with this basic discussion of the various types of references in texts. While paper print-outs certainly have a benefit of explicitly labeling notable document fragments, the digital formats take this feature to a new level, as you can now actively navigate your document by interacting with the labels. And importantly, this is not only true for web documents, but also with any current PDF document. While PDF were originally created for platform-independent but print-oriented documents, they have partially evolved to also support various structural annotations and active features, akin to HTML documents.

[6] It is not uncommon to map a MarkDown document into LaTeX, in order to use the state-of-the-art TeX algorithms for creating beautiful printable documents.

Why is this important? Because it demonstrates the direction of evolution of our digital documents: originally print-oriented formats are starting to allow the embedding of more abstract structural information. In a nutshell:

> Structural document formats, such as HTML, generalize over and may eventually supersede print-oriented formats, such as PDF.

But if you want to see them on paper today, you would have to do some legwork.

The Typesetting Iceberg

If you want to author a book as a web document and print it out, you could be in for a ride. On the surface, browsers can do a good job of interpreting a CSS stylesheet and displaying it beautifully on your screen, but they are far from being great at printing them out. Printers and screens are very different animals.

Printers **require different information** from your browser's rendering engine. And that information is very low-level. To create a printable document, we first need to know how to:

– generate a table of contents,
– determine the page- and line-breaks,
– position page numbers,
– position footnotes and other bits conveniently hidden in hover and click events,
– position floating objects, such as figures and tables
– ... the list goes on.

The invisible considerations that a typesetting engine performs behind the scenes are numerous and one should venture into that sea with caution.

There is a meaningful symmetry here when comparing to the LaTeX to HTML direction. For a vanilla set of well-behaved HTML structures, you can get a beautiful PDF that satisfies most use cases. For more advanced effects, usually involving CSS and/or JavaScript, there are again two ways out: either you forbid the use of certain techniques or dedicate developer time to develop special stylesheets.

What's good for the goose is good for its webpage

In 2015, if you are creating a manuscript with the intent of publishing it as (part of) a book or some other formal proceedings, you, or more likely your publisher, are likely to target three distinct forms of your work: a printed book, an e-Book and, more rarely, a web document. Luckily for us, the world is simple in at least one aspect, as e-Books are just web documents with a bit of extra metadata.

Why should we care about both printable and web documents? One view is that we are in the process of transitioning from a print-only to a digital-only age. Another is that neither is going away, and there are different uses for different media.[7]

But the status quo sends a clear message:

> At least for the moment, authors and their tools need to concern themselves with **both** print and web workflows.

Translating between the islands of the "paradigm shift" is far from trivial, and getting all the little bits right is difficult, but it isn't impossible. In fact, it's necessary.

[7] Fans of audio books may find this view sensible, as no one thinks we are moving to an audio-only publishing world.

Draw the Line

Having stated the premise that we need to be able to publish in both paradigms, I don't think we are at a stage where we have a clear winner. We have looked at two authoring methodologies that have the basic capabilities we are looking for, but both were leaning towards extremes that in the end are detrimental.

MarkDown has a very rewarding authoring experience, as it has a minimal set of quick-to-learn markers which can get authors to a good looking document in the matter of minutes. However, a number of fragmented efforts and communities are still struggling to draft extensions to MarkDown that would allow the full range of capabilities scientists need for scholarly publishing. Unlike LaTeX, MarkDown is not designed to be extensible, which makes extensions to the language a complex political process.

LaTeX hits the other extreme, where an author needs to reign in the unlimited potential of the typesetting system as they write, which could be both distracting and destructive. The other practical problem concerns the dated syntax and programming interface, which are at odds with modern best practices.

Whether MarkDown would evolve to support the full needs of scholarly publishing, or LaTeX would experience a renaissance that redesigns it as a web authoring syntax, or even Rich-Text engines become exceptionally good, for me the jury is still out.

You Forgot WYSIWYG!

"What You See Is What You Get" editing or WYSIWYG for short is a catch-all name for the family of visual text editors out there. This approach does away with the idea of a separate authoring syntax and instead keeps the editing flow in a constant preview of the final document. Advanced editing operations are performed via palettes of buttons and selectors.

Using Rich-Text editing on the web, or one of the Office solutions for the desktop are two variations of WYSIWYG.

Rich-Text could in principle support the full range of scholarly features, but would require a non-trivial JavaScript engine for citations and references. It would also need a new solution for creating high quality printable manuscripts.

Desktop Office solutions have their own abstract document representation and usually have good exports to web and printable documents, but they exist as standalone toolkits, rather than standardized formatting languages. Publishing a web document with a Desktop Office solution would require manually moving around a set of HTML files, or a special adapter for each web venue of interest. Also, the offline-only editing caps the ability of having an "active" document quite early, as it sets obstacles for the authoring user experience, such as collaborative editing with multiple authors. I will go into more detail in a later blog post.

Looking Forward

All of these approaches have potential, but none of them provides a completely satisfying solution at the start of 2015. However, we already know the shape and size of our desired destination:

1. Support both print and web paradigms
2. Offer a compelling, collaborative, authoring experience
3. Enable the full palette of tools for scholarly publishing
4. Embrace the domain of active, data-rich documents
5. Create beautiful scientific manuscripts

What will get us there?

Helpful Product Placement

This blog post is cross-hosted on both Authorea and a static web site. For the static hosting, I currently write content in LaTeX and have a smart bit of Ruby create a PDF and an HTML5 blog post out of my "text program", using XeLaTeX and LaTeXML. I then upload the bundle to GitHub and deploy to my Ruby on Rails site on Digital Ocean. This sounds like, and in fact is, an exercise in programming and juggling technology stacks.

If you value your free time, don't try this at home. If you're a developer who loves FLOSS and you find this article fun – write to me, there is a lot you can do!

For everyone else, there is Authorea.

Our team is passionate about Open Science and high quality 21st century manuscripts, and we want to make sure you can speak science without having to scream in code.

References

KCD+11. Michael Kohlhase, Joe Corneli, Catalin David, Deyan Ginev, Constantin Jucovschi, Andrea Kohlhase, Christoph Lange, Bogdan Matican, Stefan Mirea, and Vyacheslav Zholudev. The planetary system: Web 3.0 & active documents for stem. volume 4, pages 598–607. Elsevier, 2011. Finalist at the Executable Paper Grand Challenge.