peanuts in red sauce

webster-cli

webster-cli

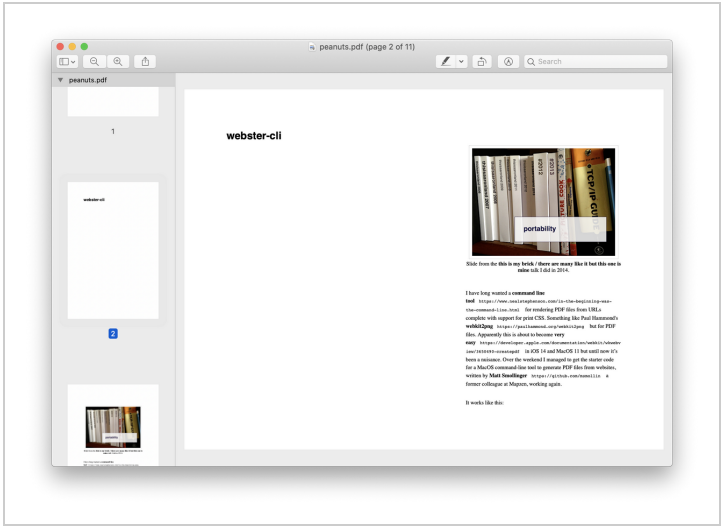
Slide from the **this is my brick / there are many like it but this one is mine** talk I did in 2014.

For a long time I have wanted a **command line tool** `https://www.nealstephenson.com/in-the-beginning-was-the-command-line.html` for rendering PDF files from URLs complete with support for **print CSS** `https://www.w3.org/community/cssprint/` . Something like Paul Hammond's **webkit2png** `https://paulhammond.org/webkit2png` but for PDF files. Apparently this is about to become **very easy** `https://developer.apple.com/documentation/webkit/wkwebview/3650490-createpdf` in iOS 14 and MacOS 11 but until now it's been a nuisance. Over the weekend I managed to get the starter code for a MacOS command-line tool to generate PDF files from websites, written by **Matt Smollinger** `https://github.com/msmollin` a former colleague at Mapzen, working again.

It works like this:

```
$> webster-cli https://aaronland.info/weblog/2020/08/24/peanuts --height 9 --width 6
```

And it produces a file that looks like this:



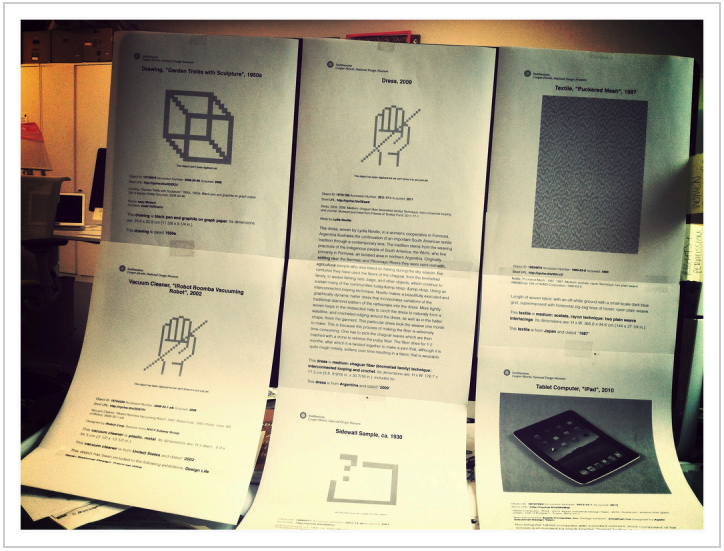
You can see the actual PDF file at **here**.

That's all it does. I was asked by a friend why this tool was necessary and this is what I said:

Mostly for making **print-on-demand-ready editions** https://www.aaronland.info/weblog/2019/01/30/something/#millsfield of personal websites so that there is a copy of that work that **doesn't require electricity** https://www.aaronland.info/weblog/2017/02/20/mostly/#museumnext to work. I've done the same for work stuff like baking a PDF copy of the **Mapzen weblog** https://www.mapzen.com/blog (and uploading it to **the Internet Archive** https://archive.org/search.php?query=mapzen%20weblog) when it wasn't clear what would survive **the shutdown of the company** https://www.aaronland.info/weblog/2017/12/31/things/#chapter-two .

The goal is never to supplant websites or plain vanilla HTML but to use them as the seed for the PDF file, hence the use of WebKit which has proper support for print CSS.

In a museum context it becomes an interesting way to think about generating custom on-demand "catalogs" for people. This is especially interesting for a place like **SFO Museum** https://millsfield.sfomuseum.org/ since people will be getting on planes with limited or garbage internet access.



Examples of objects from the **Cooper Hewitt** collection rendered using print stylesheets.

We did some work around print stylesheets for museum objects at the Cooper Hewitt. In a 2013 blog post titled "**cmd-P**" `https://labs.cooperhewitt.org/2013/cmd-p/` Katie Shelly wrote:

> What we realized in looking at all the printouts, though, is that the simplified view of a collection record resembles a gallery wall label. And we're currently knee-deep in the wall label discussion here at the Museum as we re-design the galleries (what does it need? what doesn't it need? what can it do? how can it delight? how can it inform?).

webster-cli `https://github.com/aaronland/webster-cli` is not

the first tool to generate PDF files from webpages. I used to use a tool called **wkpdf** `https://github.com/plessl/wkpdf` until it stopped being maintained. There is also the **wkhtmltopdf** `https://github.com/wkhtmltopdf/wkhtmltopdf` tool which has been around for ages. I haven't tried using it recently but in the past I've found the support for print stylesheets to be lacking and it has a long list of dependencies that make it difficult to install. Unlike these tools **webster-cli** `https://github.com/aaronland/webster-cli` is a native MacOS application. My hope is that this will give it a little more stability over the long run while also ensuring access to all the latest features when it comes to rendering web pages.

I've taken Matt's original tool, called **webster** `https://github.com/msmollin/webster`, and broken it in to two pieces:

- ***webster-cli*** `https://github.com/aaronland/webster-cli` *, the command line tool I described above.*

- ***swift-webster*** `https://github.com/aaronland/swift-webster` *, a generic Swift package that does all the work of producing PDF files.*

One of the reasons for breaking things up this way is that, while thinking about how SFO Museum might use these tools to generate a catalog of web pages on demand I realized that this is code that will only work on a MacOS or iOS device. Even though **Swift** `https://swift.org/`, the programming language used to write the tool, has been ported to Linux, and can even be used to **write Lambda functions** `https://swift.org/blog/aws-lambda-runtime/` in AWS, the **WebKit framework** `https://developer.apple.com/documentation/webkit` which handles all the work rendering web pages is only available

on operating systems produced by Apple.

In order to automate the production of PDF files it made sense to separate the code that does all the work from the code that triggers that work. That trigger might be a command line application running on my laptop or, eventually, **a web service** `https://github.com/apple/swift-nio` so that a process running on another computer can trigger the creation of a PDF file.

I mention this because one of the arguments I have been trying to advance recently is that the cultural heritage sector needs to revisit **how it produces software, and other technologies** `https://www.aaronland.info/weblog/2020/04/06/futures/#mw20` , used by and for the community. An abbreviated version of that argument is:

- *The costs of outsourcing the kinds of digital projects that cultural heritage institutions want to pursue are unsustainable. In the short-term these costs are outside the reach of most institutions so they never happen in the first place. Even for those that can afford these them, the costs associated with long-term maintenance or adapting projects to future needs are often even more expensive than the original effort.*

- *There is a widespread belief in the sector that it will never be able to attract or retain the staff needed to undertake these projects in-house. No single museum or library will ever enjoy the 400-person software engineering staffs that are so common in the private sector but there is a lot of room between 400 and "nothing". The belief that it's a binary option has become a self-fulfilling prophecy in many organizations.*

- *This belief has led to the creation of any number of consortia, typically funded by large donor organizations, to produce monolithic applications that service whole slices of the cultural heritage sector. Even with the best of motivations these initiatives almost never work as intented.*

- *Every collection practice is unique and these differences are made manifest in how they intersect with digital technologies and the demands they place on them. It is unrealistic to expect that a single umbrella organization, often operating within its own finite budget and staffing constraints, could service the wide range of needs and concerns of its members.*

- *Member organizations rarely have the technical staff on hand to discuss, at a detailed implementation level, the kinds of features and functionality needed for their specific*

> *projects so needs and requirements are only ever discussed in abstract generalities. The net result is, unsurprisingly, a lowest-common denominator solution that everyone tolerates because of the unspoken belief that there is no alternative but that no one really likes.*

In as much as there are organizations with small, or limited, technical staff I think that there needs to be a shift in the way we produce the software and tools used to service our respective digital initiatives. Although there is often a commonality of needs between two or more cultural heritage institutions there is rarely any overlap in how those needs are delivered or implemented.

In fairness the library and archives sector has had a better success, building and delivering shared tools, but the same can not be said of museums. Common tooling that aims to address the specific quirks of anything but a single museum have, historically, become an unweildly burden that no one wants to shoulder.

Instead I believe that we should undertake the effort to break apart the tools we do build in to layers of generic functionality that aren't specific a given organization or project. These **small, focused tools** `https://millsfield.sfomuseum.org/blog/2020/08/04/small-tools/` are the things that we should be sharing rather than finished projects. These should be the common layer of plumbing and a "kit of parts" that we can all reach to, that underpin the different and varied initiatives we undertake.

Implicit in this argument is the idea that there is someone in an organization to reassemble this kit of parts in concert with the specific needs and technologies bespoke to that organization. You have to believe that it's possible to have an in-house technical staff. Conversely, that in-house technical staff needs to ensure that their work is

economically viable and one way to do that is by sharing tools that can be used, and improved upon, across the cultural heritage sector.

webster-cli `https://github.com/aaronland/webster-cli` , for example, doesn't address what your print stylesheet looks like, nor does it say anything about which web pages it renders. It doesn't even handle **combining multiple PDF files** `https://pdfbox.apache.org/2.0/commandline.html#pdfmerger` in to a single "catalog". It does exactly one thing: It automates the production of a PDF file from a URL. The many different uses for that PDF file are left up to individuals using **webster-cli** `https://github.com/aaronland/webster-cli` to decide and to see through to completion.

The cultural heritage sector has never been able to afford the "400-person software" engineering staffs that are the norm in the private sector. Nor can it continue to afford the cost of third-party client-services technology providers. I believe it can, should and needs to afford something at the intersection of those two extremes and "nothing".

Fostering a practice of small, focused tools is one part of how we can make that possible.

2020-08-24