Pollen Footnotes: An Approach

Sunday, January 21st, 2018

[1] One of the things you get for free with Markdown that you have to cook from scratch with Pollen (or HTML for that matter) is footnotes. But this is fine, since it gives you more control over the results. Here is what I cooked up for use on the upcoming redesign of *The Local Yarn* weblog/book project.

The Pollen discussion group has a thread on this post[2] that is well worth reading. Matthew Butterick showed you can get mostly the same results with clearer and more concise code using normal tag functions as opposed to doing everything top-down starting with `root`.

An aside: on the web, footnotes are something of an oddity. HTML doesn't have any semantic notion of a footnote, so we typically make them up using superscripted links to an ordered list at the end of the article. I'm sympathetic to arguments that this makes for a poor reading experience, and am convinced that they are probably overused. Nonetheless, I'm converting a lot of old content that uses footnotes, and I know I'll be resorting to them in the future. Some newer treatments[3] of web footnotes use clever CSS to sprinkle them in the margins, which is nice, but comes with downsides: it isn't accessible, it's not intuitive to use and read on a phone, it renders the footnotes inline with the text in CSS-less environments (Lynx, e.g.) and the markup is screwy[4]. So I'm sticking with the old ordered-list-at-the-end approach (for this project, and for now, at least).

So I get to design my own footnote markup. Here's what's on my wishlist:

1. I want each footnote's *contents* to be defined in a separate place from the footnote *references*. This will keep the prose from getting too cluttered.

2. I want to be able to define the footnote contents anywhere in the document, in any order, and have them properly collected and ordered at the end.

3. I want to be able to use any mix of strings, symbols or numbers to reference footnotes, and have these all be converted to ordinal reference numbers.

---

1. This article assumes you are familiar with Pollen (http://pollenpub.com) and the concept of tagged X-expressions.
2. https://groups.google.com/forum/#!topic/pollenpub/laWL4SWx0Zc
3. https://edwardtufte.github.io/tufte-css/#sidenotes
4. These reasons are listed in decreasing order of importance for the particular application I have in mind.

4. I want to be able to refer to the same footnote more than once.[5] (Rare, but useful in some cases).

5. If I should happen to refer to a footnote that is never defined, I want a blank footnote to appear in the list in its place. (On the other hand if I define a footnote that isn't referenced anywhere, I'm content to let it disappear from the output.)

6. I want the footnote links not to interfere with each other when more than one footnote-using article is displayed on the same page. In other words, the URL for footnote #3 on article (A) should never be the same as the URL for footnote #3 on article (B).

In other words, I want to be able to do this:

```
Here is some ◊textfn[1]. Later on the paragraph continues.

In another paragraph, I ◊mayfn[2] refer to another footnote.◊

fndef[1]{'Heres the contents of the first footnote.}◊
fndef[2]{And here are the contents of the second one.}
```

But I also want to be able to do this:

```
◊
fndef["doodle"]{And here are the contents of the second one.}

Here is some ◊textfn["wipers"]. Later on the paragraph continues.◊
fndef["wipers"]{'Heres the contents of the first footnote.}

In another paragraph, I ◊mayfn["doodle"] refer to another footnote.
```

And both of these should render identically to:

```
<p>Here is some text<sup><a href="#550b35-1" id="550b35-1_1">1</a></sup>. Later on the paragraph
    continues.</p>

<p>In another paragraph, I may <sup><a href="#550b35-2" id="550b35_1">2</a></sup> refer to another
    footnote.</p>

<section class="footnotes"><hr />
```

5. It was this requirement in particular that steered me away from using the otherwise-excellent pollen-count (https://github.com/malcolmstill/pollen-count) package.

```
<ol>
  <li id="550b35-1">’Heres the contents of the first footnote. <a href="#550b35-1_1⍰"></a></li>
  <li id="550b35-2">And here are the contents of the second one. <a href="#550b35-2_1⍰"></a></li>
</ol>
</section>
```

6

This style of markup is a little more work to code in `pollen.rkt`, but it lets me be flexible and even a bit careless when writing the prose.

~~The output for footnotes (given my requirements) can’t very well be handled within individual tag functions; it demands a top-down approach.~~ [Again, this turns out not to be true! see the Pollen group discussion[7].] So I will be leaving my `◊fn` and `◊fndef` tag functions undefined, and instead create a single function `do-footnotes` (and several helper functions nested inside it) that will transform everything at once. I’ll call it from my `root` tag like so:

pollen.rkt

```
(require txexpr
       sugar/coerce
       openssl/md5
       pollen/decode
       pollen/template) ; ’Thats everything we need for this project

(define (root . elements)
  (define footnoted
    (do-footnotes `(root ,@elements)
              (fingerprint (first elements)))))
```

The `do-footnotes` function takes a tagged X-expression (the body of the article) and a prefix to use in all the relative links and backlinks. [8] Here are the general stages we’ll go through inside this function:

1. Go through the footnote *references*. Transform them into *reference links*, giving each an incrementally higher *reference number* (or, if the footnote has been ref-

6. You may be wondering, where did the `550b35` come from? Well, it’s an automatically generated identifier that’s (mostly, usually) unique to the current article. By using it as a prefix on our footnote links and backlinks, we prevent collisions with other footnote-using articles that may be visible on the same page. I’ll explain where it comes from at the end of this article.
7. https://groups.google.com/forum/#!topic/pollenpub/laWL4SWx0Zc
8. You may have surmised that the `fingerprint` function call above is where the `550b35` prefix came from. Again, more on that later.

erenced before, using the existing number). For later use, keep a list of all references and in the order in which they're found.

2. Split out all the footnote *definitions* from the rest of the article. Get rid of the ones that aren't referenced anywhere. Add empty ones to stand in for footnotes that are referenced but not defined.

3. Sort the footnote definitions according to the order that they are first referenced in the article.

4. Transform the footnote definitions into an ordered list with backlinks, and append them back on to the end of the article.

Here is the code for `do-footnotes` that implements the first stage:

pollen.rkt

```
(define (do-footnotes tx prefix)
  (define fnrefs '())

  (define (fn-reference tx)
    (cond
      [(and (eq? 'fn (get-tag tx))
            (not (empty? (get-elements tx))))
       (define ref (->string (first (get-elements tx))))
       (set! fnrefs (append fnrefs (list ref)))
       (let* ([ref-uri (string-append "#" prefix "-" ref)]
              [ref-sequence (number->string (count (curry string=? ref) fnrefs))]
              [ref-backlink-id (string-append prefix "-" ref "_" ref-sequence)]
              [ref-ordinal (number->string (+ 1 (index-of fnrefs ref)))]
              [ref-str (string-append "(" ref-ordinal ")")])
         `(sup (a [[href ,ref-uri] [id ,ref-backlink-id]] ,ref-str)))]
      [else tx]))

  (define tx-with-fnrefs (decode tx #:txexpr-proc fn-reference))…
  )
```

Looking at the last line in this example will help you understand the flow of control here: we can call `decode`[9] and, using the `#:txexpr-proc` keyword argument, pass it a function to apply to every X-expression tag in the article. In this case, it's a helper function we've just defined, `fn-reference`. The upshot: the body of `fn-reference` is going to be executed once for each `◊fn` tag in the article.

9. http://docs.racket-lang.org/pollen/Decode.html

By defining `fn-reference` *inside* the `do-foonotes` function, it has access to identifiers outside its scope, such as the `prefix` string but most importantly the `fnrefs` list. This means that every call to `fn-reference` will be able to check up on the results of all the other times it's been called so far. And other helper functions we'll be creating inside `do-footnotes` later on will also have easy access to the results of those calls.

So let's examine the steps taken by `fn-definition` in more detail.

1. First, using `cond` it checks to see if the current X-expression `tx` is a `fn` tag and has at least one element (the reference ID). This is necessary because `decode` is going to call `fn-reference` for *every X-expression in the article*, and we only want to operate on the ◊`fn` tags.

2. Every time `fn-reference` finds a footnote reference, it has the side-effect of appending its reference ID (in string form) to the `fnrefs` list (the `set!` function call). Again, that list is the crucial piece that allows all the function calls happening inside `do-footnotes` to coordinate with each other.

3. The function uses `let*` to set up a bunch of values for use in outputting the footnote reference link:

   (a) `ref-uri`, the relative link to the footnote at the end of the article.

   (b) `ref-sequence`, will be "`1`" if this is the first reference to this footnote, "`2`" if the second reference, etc. We get this by simply counting how many times `ref` appears in the `fnrefs` list so far.

   (c) `ref-backlink-id` uses `ref-sequence` to make an id that will be the target of a ⮌ back-link in the footnote definition.

   (d) `ref-ordinal` is the footnote number as it will appear to the reader. To find it, we remove all duplicates from the `fnrefs` list, find the index of the current `ref` in that list, and add one (since we want footnote numbers to start with 1, not 0).

   (e) `ref-str` is the text of the footnoote reference that the reader sees. It's only used because I wanted to put parentheses around the footnote number.

4. Then, in the body of the `let*` expression, the function outputs the new footnote reference link as an X-expression that will transform neatly to HTML when the document is rendered.

So after the call to `decode`, we have an X-expression, `tx-with-fnrefs`, that has all the footnote references (`◊fn` tags) properly transformed, and a list `fnrefs` containing all the footnote reference IDs in the order in which they are found in the text.

Let's take a closer look at that list. In our first simple example above, it would end up looking like this: `'("1" "2")`. In the second example, it would end up as `'("wipers" "doodle")`. In a very complicated and sloppy document, it could end up looking like `'("foo" "1" "7" "foo" "cite1" "1")`. So when processing `◊fndef["foo"]`, for example, we can see by looking at that list that this should be the first footnote in the list, and that there are two references to it in the article.

All that said, we're ready to move on to phase two through four.

pollen.rkt

```
(define (do-footnotes tx)
  ; …stage 1 above …

  (define (is-fndef? x) (and (txexpr? x) (equal? 'fndef (get-tag x))))

  ; Collect ◊fndef tags, filter out any that 'arent actually referenced
  (define-values (body fn-defs) (splitf-txexpr tx-with-fnrefs is-fndef?))
  (define fn-defs-filtered
    (filter λ((f)
            (cond
              [(member (->string (first (get-elements f))) fnrefs) #t]
              [else #f]))
          fn-defs))

  ; Get a list of all the IDs of the footnote *definitions*
  (define fn-def-ids
    (for/list ([f (in-list fn-defs-filtered)]) (->string (first (get-elements f)))))

  ; Pad the footnote definitions to include empty ones for any that 'werent defined
  (define fn-defs-padded
    (cond [(set=? fnrefs fn-def-ids) fn-defs-filtered]
        [else (append fn-defs-filtered
                  (map λ( (x) `(fndef ,x (i "Missing footnote definition")))
                      (set-subtract fnrefs fn-def-ids)))]))
  ; …stage 3 and 4 …
)
```

We define a helper function `is-fndef?` and use it with `splitf-txexpr` to extract all the `◊fndef` tags out of the article and put them in a separate list. Then we use `filter`, passing it an anonymous function that returns `#f` for any `fndef` whose ID doesn't appear in `fndefs`.

Now we need to deal with the case where the `◊fn` tags in a document reference a footnote that is never defined with an `◊fndef` tag. To test for this, we just need a list of the

reference IDs used by the footnote definitions. The definition of `fn-def-ids` provides this for us, using `for/list` to loop through all the footnote definitions and grab out a stringified copy of the first element of each. We can then check if `(set=? fnrefs fn-def-ids)`—that is, do these two lists contain all the same elements (regardless of duplicates)? If not, we use `set-subtract` to get a list of which IDs are missing from `fn-def-ids` and for each one, append another `fndef` to the filtered list of footnote definitions.

pollen.rkt

```
(define (do-footnotes tx)
  ; …stages 1 and 2 above …

  (define (footnote<? a b)
    (< (index-of (remove-duplicates fnrefs) (->string (first (get-elements a))))
       (index-of (remove-duplicates fnrefs) (->string (first (get-elements b))))))

  (define fn-defs-sorted (sort fn-defs-padded footnote<?))

  ; …stage 4 …
```

The helper function `footnote<?` compares two footnote definitions to see which one should come first in the footnote list: it compares them to see which one has the ID that appears first in `fndefs`. We pass that function to `sort`, which uses it to sort the whole list of footnote definitions.

We are almost done. We just have to transform the now-ordered list of footnote definitions and append it back onto the end of the article:

pollen.rkt

```
(define (do-footnotes tx)
  ; …stages 1 to 3 above …

  (define (fn-definition tx)
    (let* ([ref (->string (first (get-elements tx)))]
           [fn-id (string-append "#" prefix "-" ref)]
           [fn-elems (rest (get-elements tx))]
           [fn-backlinks
            (for/list ([r-seq (in-range (count (curry string=? ref) fnrefs))])
              `(a [[href ,(string-append "#" prefix "-" ref "_"
                                         (number->string (+ 1 r-seq)))]] ↩""))])
      `(li [[id ,fn-id]] ,@fn-elems ,@fn-backlinks)))

  (define footnotes-section
    `(section [[class "footnotes"]] (hr) (ol ,@(map fn-definition fn-defs-sorted))))

```

```
17   (txexpr (get-tag body)
18         (get-attrs body)
19         (append (get-elements body)
20               (list footnotes-section)))
21   ; Finis!
22 )
```

We need one more helper function, `fn-definition`, to transform an individual `◊fndef` tag into a list item with the footnote's contents and backlinks to its references. This helper uses `let*` in a way similar to `fn-reference` above, constructing each part of the list item and then pulling them all together at the end. Of these parts, `fn-backlinks` is worth examining. The expression `(curry string=? ref)` returns a function that compares any string to whater `ref` currently is.[10] That function gets passed to `count` to count how many times the current footnote is found in `fnrefs`. The list comprehension `for/list` can then use that range to make a ↩ backlink for each of them.

In defining the `footnotes-section` we `map` the helper function `fn-definition` onto each `◊fndef` tag in our sorted list, and drop them inside an X-expression matching the HTML markup we want for the footnotes section. The last statement adds this section to the end of `body` (which was the other value given to us by `splitf-txexpr` way up in stage 2), and we're done.

All that remains now is to show you where I got that `550b35` prefix from.

Ensuring unique footnote IDs

As mentioned before, I wanted to be able to give all the footnotes in an article some unique marker for use in their `id` attribute, to make sure the links for footnotes in different articles never collide with each other.

When the topic of "ensuring uniqueness" comes up it's not long before we start talking about hashes.

I could generate a random hash once for each article, but then the footnote's URI would change every time I rebuild the article, which would break any deep links people may have made to those footnotes. How often will people be deep-linking into my footnotes? Possibly never. But I would say, if you're going to put a link to some text on the web, don't make it fundamentally unstable.

10. `curry` is basically a clever way of temporarily "pre-filling" some of a function's arguments.

So we need something unique (and stable) from each article that I can use to deterministically create a unique hash for that article. An obvious candidate would be the article's title, but many of the articles on the site I'm making will not have titles.

Instead I decided to use an MD5 hash of the text of the article's first element (in practice, this will usually mean its first paragraph):

pollen.rkt

```
; Concatentate all the elements of a tagged x-expression into a single string
; (ignores attributes)
(define (txexpr->elements-string tx)
  (cond [(string? tx) tx]
        [(stringish? tx) (->string tx)]
        [(txexpr? tx)
         (apply string-append (map txexpr->elements-string (get-elements tx)))]))

(define (fingerprint tx)
  (let ([hash-str (md5 (open-input-string (txexpr->elements-string tx)))])
    (substring hash-str (- (string-length hash-str) 6))))
```

The helper function `txexpr->elements-string` will recursively drill through all the nested expressions in an X-expression, pulling out all the strings found in the *elements* of each and appending them into a single string. The `fingerprint` function then takes the MD5 hash of this string and returns just the last six characters, which are unique enough for our purposes.

If you paste the above into DrRacket (along with the `require`s at the beginnning of this post) and then run it as below, you'll see

```
> (fingerprint (txexpr->elements-string '(p "Here is some text" (fn 1) ". Later on the paragraph
    continues.")))
"550b35"
```

This now explains where we were getting the `prefix` argument in `do-footnotes`:

pollen.rkt

```
(define (root . elements)
  (define footnoted
    (do-footnotes `(root ,@elements)
                  (fingerprint (first elements)))))
```

Under this scheme, things could still break if I have two articles with exactly the same text in the first element. Also, if I ever edit the text in the first element in an article, the prefix will change (breaking any deep links that may have been made by other people). But I figure that's the place where I'm least likely to make any edits. This approach brings the risk of footnote link collision and breakage down to a very low level, wasn't difficult to implement and won't be any work to maintain.

Summary and parting thoughts

When designing the markup you'll be using, Pollen gives you unlimited flexibility. You can decide to adhere pretty closely to HTML structures in your markup (allowing your underlying code to remain simple), or you can write clever code to enable your markup do more work for you later on.

One area where I could have gotten more clever would have been error checking. For instance, I could throw an error if a footnote is defined but never referenced. I could also do more work to validate the contents of my `◊fn` and `◊fndef` tags. If I were especially error-prone and forgetful, this could save me a bit of time when adding new content to my site. For now, on this project, I've opted instead for marginally faster code…and more cryptic error messages.

I will probably use a similar approach to allow URLs in hyperlinks to be specified separately from the links themselves. Something like this:

chapter.html.pm

```
#lang pollen

For more information, see ◊a[1]{About the Author}. You can also
see ◊a[2]{his current favorite TV show}.◊

hrefs{
[1]: http://joeldueck.com
[2]: http://www.imdb.com/title/tt5834198/
}
```