Splitting Pollen tags with Racket macros

Wednesday, November 9th, 2016

This may be one of the nerdiest things I have ever written, but I know there may be three or five people who will find it useful. This post is specifically for people who are using Pollen[1] to generate content in multiple output formats[2], and who may also be using a separate build system like make[3].

Normally when targeting multiple output formats in Pollen[4], you'd write a tag function something like this:

pollen.rkt

```
; …

(define (strong . xs)
  (case (current-poly-target)
    [ltx (string-append "\\textbf{" ,@xs "}")]
    [else `(strong ,@xs)]))

; …
```

Here, everything for the `strong` tag is contained in a single tidy function that produces different output depending on the current output format. This is fine for simple projects, but not ideal for more complex ones, for a couple of reasons.

First there's the issue of tracking dependencies. Let's say every Pollen file in your project gets rendered as an HTML file *and* as part of a PDF file. Then one day you make a small change in your `pollen.rkt` file. Does this edit affect just the HTML files? Or the PDF files? Or both? Which ones now need to be rebuilt? If you're doing things as shown above, there's no straightforward way for Pollen (or `make`) to determine this; you'll have to rebuild all the output files every time.

Then there's the issue of readability. Even with two possible output formats, `pollen.rkt` gets much more difficult to read. I didn't even want to think about how hairy it would get at three or four.

I decided to address this by having each output format get its own separate `.rkt` file, containing its own definitions for each tag function, prefixed by the output format:

1. http://pollenpub.com
2. https://docs.racket-lang.org/pollen/fourth-tutorial.html
3. https://www.gnu.org/software/make/manual/html_node/Overview.html#Overview
4. http://pollenpub.com

html-tags.rkt

```
(define (html-strong attrs elements)
  `(strong ,attrs ,@elements))
```

pdf-tags.rkt

```
(define (pdf-strong attrs elements)
  (string-append "\\textbf{" ,@xs "}"))
```

That part is simple enough. But you also need a way for `pollen.rkt` to branch to one tag or the other depending on the current poly target.

To handle this part, I wrote a macro, `poly-branch-tag`, which allows you to define a tag that will automatically call a different tag function depending on the current output format. The macro is rather long, but you can view it in the `polytag.rkt`[5] file of this blog's source code at Github.

Defining tag functions with `poly-branch-tag`

To use this macro, first copy the `polytag.rkt` file[6] from this blog's source code into your project.

You then include `polytag.rkt` and declare your Pollen tags using the macro. The first argument is the tag name, optionally followed by a single required attribute and/or as any number of optional attributes with default values:

pollen.rkt

```
#lang racket
(require pollen/setup)
(require "polytag.rkt")
(require "html-tags.rkt" "pdf-tags.rkt")

; Define our poly targets as usual
(module setup racket/base
   (provide (all-defined-out))
   (define poly-targets '(html pdf)))

; Simple tag with no required or default attributes
(poly-branch-tag strong)

; Tag with a single required attribute
(poly-branch-tag link url)

```

5. https://github.com/otherjoel/thenotepad/blob/master/pollen-local/polytag.rkt
6. https://github.com/otherjoel/thenotepad/blob/master/pollen-local/polytag.rkt

```
; Tag with required attribute + some optional attrs w/defaults
(poly-branch-tag figure src (fullwidth #f) (link-url ""))
```

For every tag function declared this way, write the additional functions needed for each output type in your `(setup:poly-targets)`. E.g., for `strong` above, we would define `html-strong` and `pdf-strong` inside their respective `.rkt` files.

These tag functions should always accept exactly two arguments: a list of attributes and a list of elements. The macro will ensure that any required attribute is present and any default values are applied. Here's an example:

html-tags.rkt

```
(define (html-figure attrs elems) ; Important! Tag name must have html- prefix
  (define src (attr-val 'src attrs)) ; You can grab what you need from attrs
  (if (attr-val 'fullwidth attrs) ; (I made (attr-val) to accept boolean values in attributes)
     (make-fullwidth)))      ; [dummy example]
```

The benefits, reiterated

If you use a dependency system like `make` in your Pollen project, you now have a clear separation between output files in a particular format and the code that produces output in that format. An edit to `html-tags.rkt` will only affect HTML files. An edit to `pdf-tags.rkt` will only affect PDF files. You can see this blog's makefile[7] for a detailed example.

It's also easier to add output formats without losing your sanity. Each output format gets its own `.rkt` file where you can define your tag functions all the way up to `root`, and the logic for each output format is much easier to follow than if they were all jammed in together in one file.

Finally, I found that there's a third benefit, delightful and unintended, that comes with this approach as well: `pollen.rkt`, stripped of all function definition code, becomes essentially a very readable, self-updating cheatsheet of your project's tags. See what I mean in this blog's `pollen.rkt`[8]. This alone might almost tempt me to use `poly-branch-tag` even in projects where HTML is the only format being targeted.

7. https://github.com/otherjoel/thenotepad/blob/master/makefile
8. https://github.com/otherjoel/thenotepad/blob/master/pollen.rkt