New custom templating system emitting static HTML/CSS

I want to create a better way of templating static content. In essence, I am working on a replacement for HTML.

This will be a pre-processor (written in pure Ruby) that can emit HTML (and possibly CSS and Javascript as needed).

The underlying project is Livetext -- think of it as "markdown on steroids." See https://github.com/Hal9000/livetext -- this project is "experimental, incomplete, and possibly buggy."

Goal: Describe the layout of an entire complex web page (at a high level) in no more than 25 lines.

This will be done through the use of:
- external files
- higher-level abstractions
- custom commands and functions
- "convention over configuration"

My immediate purpose is to use this in my own custom blogging system (e.g., for blog and post templates).

I am a huge believer in hiding details and complexity. I don't believe that HTML and CSS as they exist today promote those concepts.

Broad design principles:

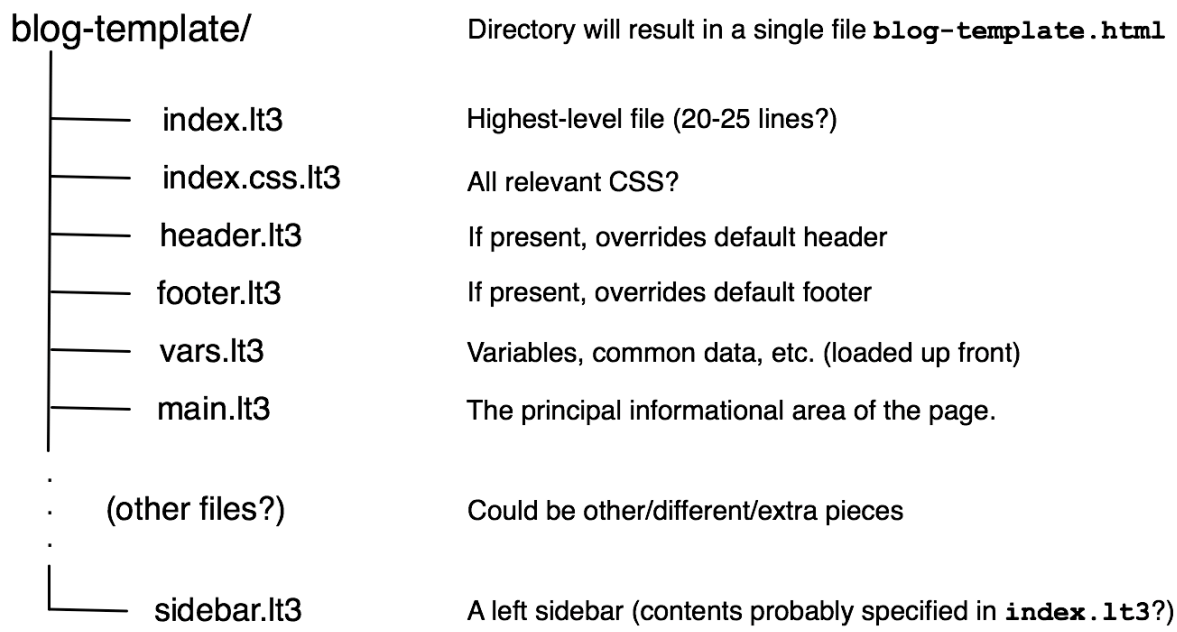
- Remember what Rails calls "convention over configuration." I haven't fully thought this through, but I like the principle.
- Let's handle the most common cases first. We'll hardcode `en_US`, for example.
- Let's make simplifying assumptions which we can refine later as needed.
- Make 90% of the cases easier, even if 9% are unchanged and 1% are harder.
- Remember the DRY principle - "Don't Repeat Yourself." For example: If `title` is used in five places, we should only set it once.
- HTML files have tended to be *very long and deeply nested*. Think about *how and why* they got that way. You wouldn't write a method with 1,000 lines in it,

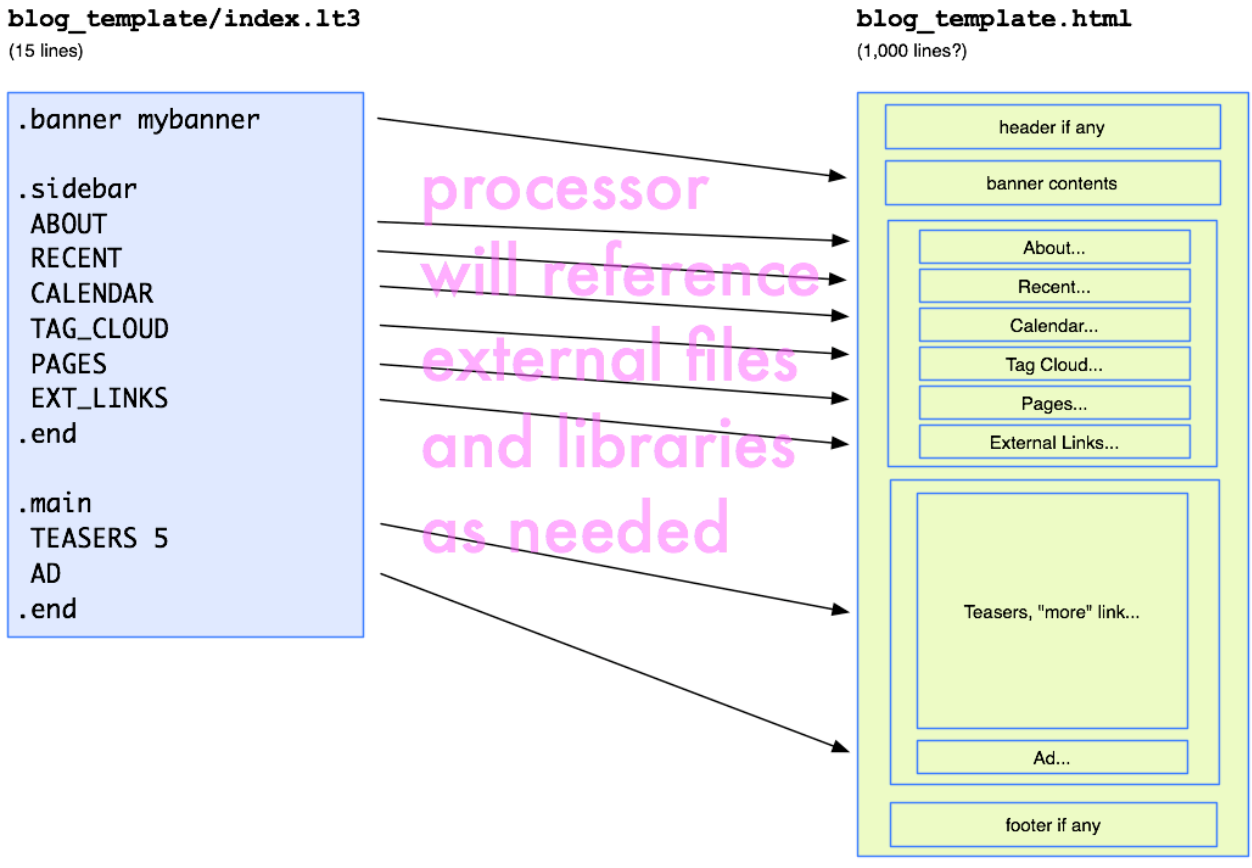
would you? But HTML was created without *variables* and without any kind of "include" feature.

- But Livetext does have variables and multiple kinds of file inclusion. It also has many other "dot commands" and several *functions* defined (and there should be many more later).
- Note that the user can define not only variables, but his own dot commands and functions as well. This is potentially *powerful*.
- Therefore let's break things down into *many small pieces*. (Think of the people who want to put crazy limits on number of lines in a method.)
- Another principle I want to use is: If a piece of text is *always there*, then omit it and let the processor insert it based on context.
- A similar principle has to do with "parameters" in pretty much any kind of context. There are three parts:
    1. If text or data is *always* the same, it's not a parameter -- hide it as deeply as you can.
    2. If it's *very often* the same, it needs a reasonable default -- set one internally and let us omit it at higher levels.
    3. If it varies often and can have many possible values -- it's a “real" parameter.
- I'd like to push this as far as I reasonably can. **Example**: Given a file that is going to be a complete HTML document: Don't put **`html`** and **`body`** tags at the top and bottom -- build an environment where the coder and processor recognize these files from context. Let the processor itself add those on. (Analogy: It is possible to insert assembly language code into a C program... but we only do it rarely and when we have good reason.)

Hypothetical example: Refer to these images. A 15-line template gets converted into potentially hundreds of lines.

An imaginary "blog template" directory structure





My Blog...

Insert clever subtitle here or whatever

About
Contact, ...

Recent
It's Rainin' Code Out Thar
Blah blah blah..
More of the same
It it gonna rain code?
More...

Calendar

Tag cloud

Pages

Ext Links

It's raining code out there... May 21, 2020

Grab a bitbucket and catch all you can. Lorem ipsum dolor, y'all.

Read this blog entry, or else May 17, 2020

They say that the answer to life, the universe, and everything is 42. But what if you take the square root of it and divide by pi?

This is also a blog entry May 15, 2020

Do you believe me? Or am I going to have to hire independent investigators to do a bunch of double-blind studies and statistical analysis?

This is a blog entry May 14, 2020

Surprise, surprise.

Alligators and stuff May 12, 2020

Some alligators, they say, can grow up to 15 feet. But a recent study by the National Science Foundation suggests that most have only four.

Human beings May 10, 2020

They're kind of overrated, and they can be jerks at times. But at the end of the day, what other species is going to sit and drink coffee with you?

Sick and tired May 8, 2020

People everywhere are being told that they're sick and tired. But are they really? I'm not, and I'm sick and tired of being told that I am.

More...

We're there when you need us. I'm lovin' it!
Taste the rainbow. Easy, breezy, beautiful...
Just Do It. Think Different. Finger-lickin' good.

My Blog...

Insert clever subtitle here or whatever

About
Contact, ...

Recent
It's Rainin' Code Out Thar
Blah blah blah..
More of the same
It it gonna rain code?
More...

Calendar

Tag cloud

Pages

Ext Links

May 21, 2020

It's Raining Code Out There...

Grab a bitbucket and catch all you can. Lorem ipsum dolor, y'all. Lorem ipsum dolor blah blah blah Lorem ipsum dolor blah blah blah Lorem ipsum dolor blah blah blah Lorem ipsum dolor blah blah blah Lorem ipsum dolor blah blah blah...

Lorem ipsum dolor blah blah blah Lorem ipsum dolor blah blah blah Lorem ipsum dolor blah blah blah Lorem ipsum dolor blah blah blah Lorem ipsum dolor blah blah blah...

Lorem ipsum dolor blah blah blah Lorem ipsum dolor blah blah blah Lorem ipsum dolor blah blah blah Lorem ipsum dolor blah blah blah Lorem ipsum dolor blah blah blah...

And in conclusion, it's rainin' code out there 'cause I said so. Go listen to my TED Talk, yo.

<< Back

When engineers try to do marketing:

"Would you be interested in ingesting a cold sugary brown liquid, given anecdotal evidence from others that the experience was a positive one?"

What are some Livetext features? See list below. See also the repo at `https://github.com/Hal9000/livetext`

A few Livetext features:

1. In general, a line of ordinary text is passed through unchanged.
2. Any line starting with a dot and a command name is a call to that command (which may be predefined or user-defined).
3. Any line starting with whitespace followed by a dollar sign and a dot command is just like an idented version of that command.
4. A dot command may reference whitespace-separated arguments; it may also access the remainder of the line as a single string (including any whitespace which is passed through as-is).
5. Some dot commands expect a "body" of text after the command; this is terminated by a .end command. A command may have a body or a set of parameters or both.
6. Set variables by **`.set name=value`** and reference by **`$name`** anywhere.
7. Functions (predefined or user-defined) are referenced by **`$$name`** and may have a string parameter in brackets.
8. The **`.include`** command will read a file and interpret it as Livetext before inserting it.
9. The **`.copy`** command will insert a file without interpretation.
10. The **`.mixin`** command will load a file of Ruby code into the namespace, with each method becoming a dot command.
11. The **`.def`** and **`.func`** commands can be used inside a Livetext file to create dot commands and functions.
12. A line starting with dot and space is ignored as a comment.