Gigamonkeys Markup

This library is based on the code I used while writing *Practical Common Lisp* to generate both PDFs (which I used for my own proofreading purposes) and the HTML version of the book that is available on the web at `http://www.gigamonkeys.com/book/`.

I had two goals:

- To be able to work in Emacs yet not have to embed an incredible amount of formatting gunk in my text while I worked on it.
- To be able to generate output in multiple forms, initially PDF and HTML and later (when it came time to turn stuff in to Apress) RTF.

In the end I settled on a mix of TeX-style markup (such as \i{...} to indicate italic) and various kinds of Wiki-inspired markup such as indicating new paragraphs by separating them with blank lines and inferring the style of certain paragraphs from their indentation. And because part of the point was to be able to write in Emacs I decided to make my markup language grok Emacs's outline-mode headers, lines prefixed by a number of `*`'s. I found this to be a pleasing format for writing

Parsing

The code here is a somewhat cleaned up version of the code I actually used while working on the book. The markup language itself is implemented in `parser.lisp`. The function `PARSE-FILE` takes a pathname designator and returns a list of s-expressions representing the parse tree. For instance if the file `sample.txt` contained the following:

```
* A Sample
Here is a paragraph. Here's a sentence with some \i{formatting} in
it. And another sentence.
And this is another paragraph.
```

then `(parse-file "sample.txt")` would return this:

```
((:H1 "A Sample")
 (:P "Here is a paragraph. Here's a sentence with some "
  (:I "formatting") " in" #\Newline "it. And another sentence.")
 (:P "And this is another paragraph." #\Newline))
```

The `:H1` tag is derived from the number of asterisks leading the line `"A Sample"`; the `:P` tags are inferred from the lack of any other tag; and the `:I` tag is taken from the name between the `\` and the `{`.

For the most part, the parser doesn't care about the names used as tag names in the `\`*name*`{...}` syntax. However it does need to know whether a given tag name is the name of paragraph tag (which can appear where an untagged paragraph can) or a subdocument tag (which is parsed like the document as a whole and thus can contain paragraphs). It recognizes tags as being one of these two types using the variables `*PARAGRAPH-TAGS*` and `*SUBDOCUMENT-TAGS*` which are defined in `parameters.lisp` and can be bound or set as you see fit.[1]

Rendering

The files `pdf.lisp` and `html.lisp` each provide implementations of the generic function `RENDER-AS` which takes an argument indicating the type of output, (either `:pdf` or `:html`), an s-expression as returned by `PARSE-FILE`, and the name of the output file. The public interface to `RENDER-AS` is the function `RENDER` which takes a an output type indicator, the name of an input file, and, optionally, the name of an output file which is otherwise derived from the name of the input file with the the output type as the file extension. Thus this documentation which is in the file `docs.txt` is rendered into PDF as:

```
MARKUP> (render :pdf "docs.txt")
#P"/Users/peter/projects/lisp-libraries/development/markup/docs.pdf"
```

New output formats can be defined simply by defining a new method on `RENDER-AS` specialized on a different output type.

Possible improvements

Styles The current PDF rendering hard-wires assumptions about how to render different document elements. And the HTML rendering simply provides a way to map from document elements to HTML tags (possibly with attributes), defaulting to mapping to the HTML tag of the same name. Ideally there'd be a way to express style information and define a mapping from the markup elements to the rendering style without changing the code. Even more ideally it'd be possible to reuse that same style information with different back-ends. Which I have some ideas about but which I've not done anything about yet.

Footnotes In both the PDF and HTML renderer, footnotes are generated as end-notes. This is probably the best you can do for HTML but in the PDF renderer it'd be nice to be able to place notes as true footnotes. However this is a non-trivial problem because making room for a footnote at the bottom of a page affects what body text fits on the page which can affect what footnotes should appear on that page. In the worst case scenario putting a footnote on a page causes the text containing the footnote reference to no longer fit on the page meaning the footnote shouldn't go on that page after all. So you take it off but now the footnote reference is back on the page. And so on.

Other output formats When working on the book I implemented an incredibly hacky RTF generator that allowed me to produce RTF documents with all the Apress styles appropriately applied since I ultimately had to turn in Word files to Apress. I'd like to clean that up and add it back in in some form because there's nothing better than not having to use Microsoft Word.

Installation

In theory all you need to do to use this library are grab the distribution, unpack it, and arrange for ASDF to know about the `.asd` file, either by adding a symlink to it from a directory that is already in `asdf:*central-registry*` or by adding the directory created when you unpack the distribution to `asdf:*central-registry*`. You'll also need my FOO library (which is documented in Chapters 30 and 31 of *Practical Common Lisp* and a couple libraries it depends on. All these are also loadable using ASDF. And, of course, to render PDF files you'll need Marc Battyani's excellent CL-PDF and CL-TYPESETTING libraries. Those libraries are available here: