kitabu

This guide will help you understand how all of the pieces fit together on Kitabu.

BY NANDO VIEIRA

THIS PAGE INTENTIONALLY LEFT BLANK

Contents

CHAPTER 1

Getting Started

This guide is designed for beginners who want to get started with Kitabu from scratch. However, to get the most out of it, you need to have some prerequisites installed:

- The Ruby interpreter version 2.0.0 or greater.
- The PrinceXML converter version 9.0 or greater.
- The KindleGen converter.

Installing Ruby

To install Ruby, consider using RVM or rbenv, both available for Mac OSX and Linux distros. If you're running a Windows, well, I can't help you. I don't even know if Kitabu runs over Windows boxes, so if you find any bugs, make sure you let me know.

Installing PrinceXML

PrinceXML is the best HTML to PDF converter available. You can use advanced CSS features to style your book in any way you want. But good things don't come for free, and PrinceXML is no exception. The Professional License, which you grant you a installation on

a single computer by a single user costs 495USD. If you don't like the price tag, consider using DocRaptor when you're ready to publish your book.

To install PrinceXML, go to the website and download the correct version for your platform; you can choose from Mac OSX, to Linux and Windows.

Installing KindleGen

KindleGen is the command-line tool that allows you to convert e-pubs into `.mobi` files. You can't sell these files, though.[1] So if that's the case, consider using Calibre for this task.[2]

If you're running Homebrew on the Mac OSX, you can install it with `brew install kindlegen`. Go to KindleGen's website and download the appropriate installer otherwise.

---

1 You can, but that would be a violation of Amazon's terms of use.

2 Calibre is not perfect, but does a good job.

CHAPTER 2

Creating Chapters

You can create chapters by having multiple files or directories. They're alphabetically sorted, so make sure you use a prefixed file name like `01_Introduction.md` as the file name.

If you're going to write a long book, make sure you use the directory organization. This way you can have smaller text files, which will be easier to read and change as you go. A file structure suggestion for a book about Ruby on Rails would be:

```
getting-started-with-rails
├── text
    └── 01_Guide_Assumptions.md
    └── 02_Whats_Rails.md
    └── 03_Creating_A_New_Project
        └── 01_Installing_Rails.md
        └── 02_Creating_The_Blog_Application.md
    └── 04_Hello_Rails
        └── 01_Starting_Up_The_Web_Server.md
        └── 02_Say_Hello_Rails.md
        └── 03_Setting_The_Application_Home_Page.md
    └── ...
```

Notice that the file name does not need to be readable, but it will make your life easier.

CHAPTER 3

Syntax Highlighting

What about the syntax

Kitabu uses Route as the syntax highlight formatter. It emits an output compatible with stylesheets designed for pygments, the Python library used by many.

To highlight a code block, use the fenced block syntax. The following example would be formatted as Ruby.

````
```ruby
class User
  attr_accessor :name, :email

  def initialize(name, email)
    @name = name
    @email = email
  end
end
```
````

The output would be something like this:

```ruby
class User
  attr_accessor :name, :email

  def initialize(name, email)
    @name = name
    @email = email
  end
end
```

If you're using Sublime Text, make sure you install the Markdown Extended plugin; it enables code syntax highlighting on your Markdown files.

You can also provide inline options such as line numbers and inline rendering.

````
```ruby?line_numbers=1
class User
  attr_accessor :name, :email

  def initialize(name, email)
    @name = name
    @email = email
  end
end
```
````

This would be rendered like this:

```ruby
1  class User
2    attr_accessor :name, :email
```

```
3
4     def initialize(name, email)
5       @name = name
6       @email = email
7     end
8   end
```

Lexers

Rouge comes with dozens of lexers. Check out this list, generated dynamically when you export your e-book.

Apache `apache`
configuration files for Apache web server

AppleScript `applescript`
The AppleScript scripting language by Apple Inc. (http://developer.apple.com/applescript/)

C `c`
The C programming language

Clojure `clojure`
The Clojure programming language (clojure.org)

CoffeeScript `coffeescript`
The Coffeescript programming language (coffeescript.org)

Common Lisp `common_lisp`
The Common Lisp variant of Lisp (common-lisp.net)

Config File `conf`
A generic lexer for configuration files

C++ `cpp`
The C++ programming language

C# `csharp`
a multi-paradigm language targeting .NET

CSS `css`
Cascading Style Sheets, used to style web pages

Dart `dart`
The Dart programming language (dartlang.com)

diff `diff`
Lexes unified diffs or patches

Elixir `elixir`
Elixir language (elixir-lang.org)

ERB `erb`
Embedded ruby template files

Erlang `erlang`
The Erlang programming language (erlang.org)

Factor `factor`
Factor, the practical stack language (factorcode.org)

Gherkin `gherkin`
A business-readable spec DSL (github.com/cucumber/cucumber/wiki/Gherkin)

Go `go`
The Go programming language (http://golang.org)

Groovy `groovy`
The Groovy programming language (groovy.codehaus.org)

Haml `haml`
The Haml templating system for Ruby (haml.info)

Handlebars `handlebars`
the Handlebars and Mustache templating languages

Haskell `haskell`
The Haskell programming language (haskell.org)

HTML `html`
HTML, the markup language of the web

HTTP `http`
http requests and responses

INI `ini`
the INI configuration format

Io `io`
The IO programming language (http://iolanguage.com)

Java `java`
The Java programming language (java.com)

JavaScript `javascript`
JavaScript, the browser scripting language

Json `json`
JavaScript Object Notation (json.org)

Liquid `liquid`
Liquid is a templating engine for Ruby (liquidmarkup.org)

Literate CoffeeScript `literate_coffeescript`
Literate coffeescript

Literate Haskell `literate_haskell`
Literate haskell

LLVM `llvm`
The LLVM Compiler Infrastructure (http://llvm.org/)

Lua `lua`
Lua (http://www.lua.org)

Make `make`
Makefile syntax

Markdown `markdown`
Markdown, a light-weight markup language for authors

MATLAB `matlab`
Matlab

MoonScript `moonscript`
Moonscript (http://www.moonscript.org)

nginx `nginx`
configuration files for the nginx web server (nginx.org)

Nim `nim`
The Nim programming language (http://nim-lang.org/)

Objective-C `objective_c`
an extension of C commonly used to write Apple software

OCaml `ocaml`
Objective CAML (ocaml.org)

Perl `perl`
The Perl scripting language (perl.org)

PHP `php`
The PHP scripting language (php.net)

Plain Text `plaintext`
A boring lexer that doesn't highlight anything

Prolog `prolog`
The Prolog programming language (http://en.wikipedia.org/wiki/Prolog)

.properties `properties`
.properties config files for Java

Puppet `puppet`
The Puppet configuration management language (puppetlabs.org)

Python `python`
The Python programming language (python.org)

QML `qml`
QML, a UI markup language

R `r`
The R statistics language (r-project.org)

Racket `racket`
Racket is a Lisp descended from Scheme (racket-lang.org)

Ruby `ruby`
The Ruby programming language (ruby-lang.org)

Rust `rust`
The Rust programming language (rust-lang.org)

Sass `sass`
The Sass stylesheet language language (sass-lang.com)

Scala `scala`
The Scala programming language (scala-lang.org)

Scheme `scheme`
The Scheme variant of Lisp

SCSS `scss`
SCSS stylesheets (sass-lang.com)

sed `sed`
sed, the ultimate stream editor

shell `shell`
Various shell languages, including sh and bash

Slim `slim`
The Slim template language

Smalltalk `smalltalk`
The Smalltalk programming language

SML `sml`
Standard ML

SQL `sql`
Structured Query Language, for relational databases

Swift `swift`
Multi paradigm, compiled programming language developed by Apple for iOS and OS X development. (developer.apple.com/swift)

Tcl `tcl`
The Tool Command Language (tcl.tk)

TeX `tex`
The TeX typesetting system

TOML `toml`
the TOML configuration format (https://github.com/mojombo/toml)

Visual Basic `vb`
Visual Basic

VimL `viml`
VimL, the scripting language for the Vim editor (vim.org)

XML `xml`
XML

YAML `yaml`
Yaml Ain't Markup Language (yaml.org)

And if what you want is not on this list, make you open a ticket on the project.

CHAPTER 4

Dynamic Content

Sometimes you may find useful to generate content dynamically. Maybe you're going to read some configuration file, or maybe you just want to define some helpers. Kitabu has support for ERb files; all you need to do is naming your text file as `.erb`.

On the previous chapter, we listed all supported Rouge lexers. To do that, I created a helper that looks like this:

```
module Kitabu
  module Helpers
    def lexers_list
      buffer = '<ul class="lexers">'

      Rouge::Lexers.constants.each do |const|
        lexer = Rouge::Lexers.const_get(const)

        begin
          title = lexer.title
          tag = lexer.tag
          description = lexer.desc
        rescue Exception => e
          next
        end
```

```
        buffer << '<li>'
        buffer << "<strong>#{title}</strong> "
        buffer << "<code>#{tag}</code><br>"
        buffer << "<span>#{description}</span>"
        buffer << '</li>'
      end

      buffer << '</ul>'
      buffer
    end
  end
end
```

To use it, I just needed to add `<%= lexers_list %>` to my text file. This allows you to create anything you need!

Kitabu comes with some built-in helpers, such as `note`. With this helper, you can create a note that generates a HTML structure, so you can easily style it. The syntax for using the `note` helper is `note(type, &block)`.

```
<% note do %>
  Some text that will be parsed as Markdown.
<% end %>
```

By default, this will generate a `<div class="note info">` tag, but you can use anything you want.

```
<% note :warning do %>
  Some text that will be parsed as Markdown.
<% end %>
```

Check out the source for a sample on how to create block helpers like `note`.

Escaping ERb code

If you want to write a book about Rails, you're likely to use lots of ERb tags. In this case, make sure you escape the `and` markers as `<% %>` and `<%= %>`; otherwise you'll have a syntax error.

```
<%%= Date.today %>
```

CHAPTER 5

Exporting Files

You can generate files as you go. Just execute `kitabu export` from your book's root directory.

```
$ kitabu export
** e-book has been exported
```

This command will generate all supported formats[3]. The generated files will be placed on your `output` directory; the following output list only the relevant files.

```
$ tree output
output
├── images
│   ├── kitabu.png
│   └── kitabu.svg
├── kitabu.epub
├── kitabu.html
├── kitabu.mobi
├── kitabu.pdf
```

---

[1] Depend on Prince, html2text and KindleGen being available on your `$PATH`.

```
├── kitabu.print.pdf
├── kitabu.txt
└── styles
    ├── epub.css
    ├── html.css
    ├── pdf.css
    └── print.css
```

This can take a while depending on your book size, but usually the process is pretty fast. If you want to generate a specific format faster, provide the `--only` flag.

```
$ kitabu export --only pdf
```

You can also automatically generate files when something changes. You can use Guard for this, and Kitabu even generates a sample file for you. All you have to do is running `bundle exec guard`.

```
$ bundle exec guard
20:38:10 - INFO - Guard is now watching at '/Users/fnando/Projects/kitabu/examples/k
** e-book has been exported
```

Exporting PDF with DocRaptor

After exporting your files (you can use `--only pdf` for this), upload files to somewhere public, possibly your Dropbox account. You can even use curl; since the command is quite long, you can view it at https://gist.github.com/fnando/de555a08e7aab14a661a.

THIS PAGE INTENTIONALLY LEFT BLANK

Kitabu

This guide will help you understand how all of the pieces fit together on Kitabu.

Nando Vieira