

# Prawn4book

## Manuel

---

### Prawn4book<br />Manuel

- Initiation d'un livre
- Ajouter un livre à une collection
- Initiation d'une nouvelle collection
- Construction du PDF du livre
  - Forcer le re-parsing du texte
  - Pour enregistrer le fichier `texte.yaml`
  - Ouvrir le fichier PDF produit
- Ouverture du PDF
  - Construction d'un index
- Texte du livre
  - Modifier l'aspect visuel du texte dans Sublime Text
- Pages du livre
  - Les marges
- Les paragraphes
  - Les différents types de paragraphe
  - Formatage personnalisé des paragraphes ( `formater.rb` )
  - Parsing personnalisé des paragraphes ( `parser.rb` )
- Recette du livre
  - Création de la recette du livre
  - Définition des fontes
- Entête et pied de page
  - Disposition
  - Variables
- Annexe
- Ne pas afficher les espaces insécables
- Package Sublime Text
  - Choix d'une autre police
- Prawn
  - Blocs de texte avec Prawn

**Prawn4book** est une application en ligne de commande permettant de transformer un texte en PDF prêt pour l'impression, grâce au (lovely) gem `Prawn`.

Tous les exemples de ce manuel présupposent qu'un alias de la commande a été créé, grâce à :

```
> ln -s /Users/me/Programmes/Prawn4book/prawn4book.rb /usr/local/bin/prawn-for-book
```

Noter ci-dessus que la commande sera `prawn-for-book` (qui est plus simple à taper)

## Initiation d'un livre

---

- Créer un dossier dans lequel seront mis tous les éléments du livre,
  - ouvrir une fenêtre Terminal dans ce dossier,
  - jouer la commande `$> prawn-for-book init`,
  - choisir de construire un nouveau livre,
  - suivre le processus proposé et choisi.
- 

## Ajouter un livre à une collection

---

Suivre la [procédure d'initiation d'un nouveau livre](#) mais en ouvrant le Terminal au dossier de la collection (ou au dossier du livre créé dans le dossier de cette collection).

---

## Initiation d'une nouvelle collection

---

- Créer le dossier dans lequel doit être placée la collection,
  - ouvrir une fenêtre Terminal à ce dossier,
  - jouer la commande `$> prawn-for-book init`,
  - choisir de construire une collection.
- 

## Construction du PDF du livre

---

Pour lancer la fabrication du PDF qui servira à l'impression du livre, jouer la commande :

```
> cd path/to/book/folder
> prawn-for-book build
```

Certaines options permettent de travailler le livre avant sa fabrication définitive :

Pour s'arrêter à une page préciser, par exemple la 24e si on veut faire des essais minimum avec KDP :

```
$> prawn-for-book build -last=24
```

## Forcer le re-parsing du texte

Par défaut, l'application utilisera le fichier `texte.yaml` s'il existe. Pour détruire ce texte, ce qui provoquera le reparsing du texte original, ajouter l'option `--force`.

```
$> prawn-for-book build --force
```

## Pour enregistrer le fichier `texte.yaml`

Si l'on est sûr de ne plus devoir parser le texte à chaque fois et qu'on veut préciser les choses au niveau de l'impression, on peut ajouter l'option `-save` (ou `--save`) à la commande `build` :

```
$> prawn-for-book build -save
```

Cela provoquera une version *Prawn4book* du texte qui permettra de préciser le formatage.

## Ouvrir le fichier PDF produit

Pour ouvrir le document PDF à la fin de la fabrication, ajouter l'option `--open`.

```
$> prawn-for-book build --open
```

---

## Ouverture du PDF

On peut ouvrir le PDF du livre dans Aperçu à l'aide de la commande :

```
$> prawn-for-book open book
```

---

## Construction d'un index

cf. [Parsing personnalisé du texte](#) pour savoir comment parser les paragraphes pour en tirer les informations importantes.

Il s'agit donc, ici, de programmer la méthode `__paragraph_parser` pour qu'elle récupère les mots à indexer. Par exemple, si ces mots sont repérés par la balise `index:mot` ou `index:(groupe de mot)`, il suffit de faire :

```
def __paragraph_parser(paragraph)

  @table_index ||= {}
  paragraph.text.scan(/index[:\(\)(.+?)\]\?/).each do |idiom|
    @table_index.key?(idiom[0]) || @table_index.merge!(idiom[0] => [])
    @table_index[idiom[0]] << {text: idiom, parag: paragraph}
  end
end
```

À l'issue du traitement, la table `@table_index` (de l'instance `PdfBook`) contiendra en clé tous les mots trouvés et en valeur une liste de toutes les itérations.

On peut donc faire ensuite :

```
module Prawn4book
  class PdfBook
    attr_reader :table_index
  end
end
```

```

end

module ParserParagraphModule
  def __paragraph_parser(paragraph)
    #... cf ci-dessus
  end
end

module PrawnCustomBuilderModule
  def __custom_builder(pdfbook, pdf)
    pdfbook.table_index.each do |idiom, occurrences|
      pdf.text "Index de '#{idiom}'"
      pdf.text occurrences.map {|oc| oc[:parag].numero }.uniq.join(', ')
    end
  end
end
end

```

## Texte du livre

On peut travailler le texte du livre dans n'importe quel éditeur simple. [Sublime Text](#) est mon premier choix pour le moment. Notamment parce qu'il offre tous les avantages des éditeurs de code, à commencer par l'édition puissante et la colorisation syntaxique. Il suffit que le texte se termine par `.p4b.txt` ou `.p4b.md` pour que Sublime Text applique le format *Prawn4Book*.

Ce package est défini dans le dossier package `Prawn4Book` de Sublime Text. On peut ouvrir ce package rapidement en jouant :

```
$> prawn-for-book open package-st
```

## Modifier l'aspect visuel du texte dans Sublime Text

Pour modifier l'aspect du texte, il faut ouvrir le package dans *Sublime Text* (

```
$> prawn-for-book open package-st
```

) et modifier le code dans le fichier

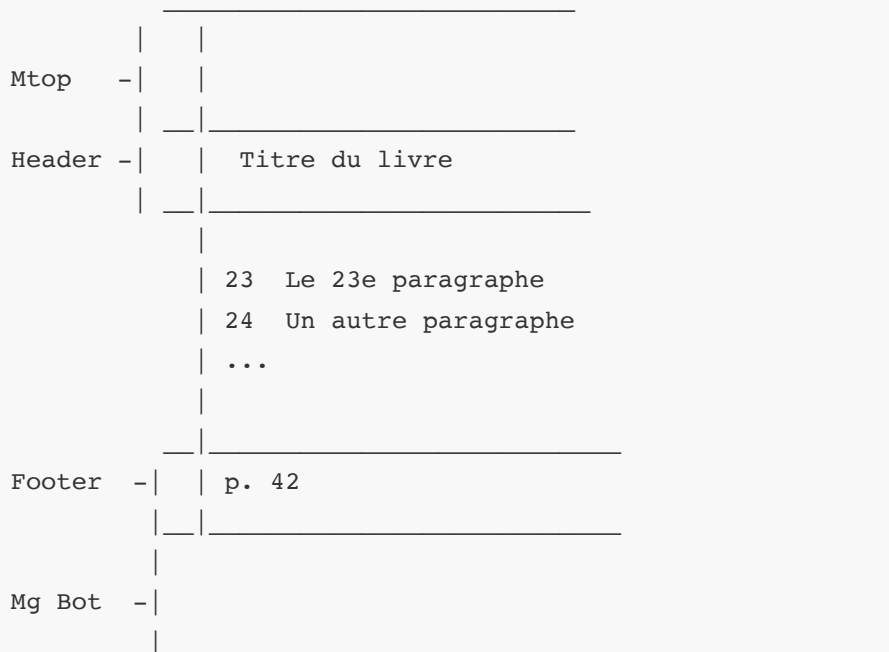
`Prawn4Book.sublime-settings` (pour la police, la taille de police, etc.) ou le fichier `Prawn4Book.sublime-color-scheme` (pour modifier la colorisation syntaxique ou les scopes).

## Pages du livre

### Les marges

Les marges sont définies de façon très strictes et concernent vraiment la partie de la page **où ne sera rien écrit**, ni pied de page ni entête. On peut représenter les choses ainsi :

v----- marge gauche (ou intérieure)



Ce qui signifie que le haut et le bas du texte sont calculés en fonction des marges et des header et footer.

## Les paragraphes

### Les différents types de paragraphe

#### Simple paragraphe

Définit dans le texte par un texte ne répondant pas aux critères suivants. Un paragraphe peut commencer par autant de balises que nécessaire pour spécifier les choses. Par exemple :  
citation:bold:center: Une citation qui doit être centrée.

#### image

Définit dans le texte par 'IMAGE[<data>]'  
Comme pour les simples paragraphes elle peut être précédée par des étiquettes de définition.

#### titre

Définit dans le texte par '#[#[#]] Titre'

## Formatage personnalisé des paragraphes ( `formater.rb` )

Le principe est le suivant :

SI un paragraphe commence par une balise (un mot suivi sans espace par '::')  
par exemple : "custag:: Le texte du paragraphe."

ALORS ce paragraphe sera mis en forme à l'aide d'une méthode de nom :

`__formate_<nom balise>`

par exemple : `def __formate_custag(string)`

QUI SERA DÉFINIE dans le fichier 'formater.rb' définissant le module  
'FormaterParagraphModule'

```
module FormaterParagraphModule # Ce nom est absolument à respecter
  def __formate_custag(string)
    # ...
    return string_formatted
  end
end
```

Ce code doit être placé dans un fichier `formater.rb` soit dans le dossier du livre soit dans le dossier de la collection si le livre appartient à une collection.

Noter que si collection et livre contient ce fichier, seul celui de la collection sera chargé.

## Parsing personnalisé des paragraphes ( `parser.rb` )

De la même manière que les paragraphes sont formatés (cf. ci-dessus), ils peuvent être parsés pour en tirer des informations utiles (pour faire un index, une bibliographie, etc.)

Il suffit pour cela de créer un fichier de nom `parser.rb` dans le dossier du livre (ou de la collection) qui contienne :

```
module ParserParagraphModule # ce nom est absolument à respecter
  def __paragraph_parser(paragraphe)
    # Parse le paragraphe {PdfBook::NTextParagraph}
    str = paragraphe.text
  end
  # ...
end #/module

module PrawnCustomBuilderModule # ce nom est absolument à respecter
  #
  # Ici doit être défini les choses à faire avec les informations
  # qui ont été parsées
```

```
#
def __custom_builder(pdfbook, pdf)
  #
  # P.e. pour insérer une nouvelle page avec du texte
  #
  pdf.start_new_page
  pdf.text "Ceci est un texte avec les infos parsées."

end
end #/module
```

Pour réaliser le texte des nouvelles pages, cf. [blocs de texte avec Prawn](#).

Ce fichier contient donc deux modules :

- **ParserParagraphModule** définit la méthode `__paragraph_parser` qui parse les paragraphes.
- **PrawnCustomBuilderModule** définit la méthode `__custom_builder` qui construit les éléments du livre en rapport avec les informations relevées.

## Recette du livre

### Création de la recette du livre

On peut créer de façon assistée la recette d'un livre en ouvrant un Terminal dans le dossier où doit être initié le livre — ou le dossier où se trouve déjà le texte, appelé `texte.txt` ou `texte.md` — et en jouant la commande : `> prawn-for-book init`.

Cette commande permet de créer un fichier `recipe.yaml` contenant la recette du livre.

### Définition des fontes

On peut être assister pour la création de la donnée des fontes (qui nécessite de connaître les chemins d'accès à toutes les fontes possibles) de cette manière :

- ouvrir un Terminal au dossier du livre ou de la collection
- jouer la commande `$> prawn-for-book aide fontes`.

```
:fonts:
  <nom utilisé>
    :<style>: "/path/to/font.ttf"
    :<style>: "/path/to/font.ttf"

# etc.
```

Par exemple :

```
# ...
```

```
dossier_fonts: &dosfonts "/Users/philippeperret/Library/Fonts"
:fonts:
  Garamond:
    :normal: "*dosfonts/ITC - ITC Garamond Std Light Condensed.ttf"
    :italic: "/Users/philippeperret/Library/Fonts/ITC - ITC Garamond Std Light
Condensed Italic.ttf"
  Bangla:
    :normal: "/System/Library/Fonts/Supplemental/Bangla MN.ttc"
    :bold:   "/System/Library/Fonts/Supplemental/Bangla MN.ttc"
  Avenir:
    :normal: "/System/Library/Fonts/Avenir Next Condensed.ttc"
  Arial:
    :normal: "/Users/philippeperret/Library/Fonts/Arial Narrow.ttf"
```

## Entête et pied de page

On peut définir les entêtes et les pieds de page dans le fichier recette du livre ou de la collection grâce aux données `:headers` et `:footers`.

Noter qu'ils sont au pluriel

Le principe est que pour chaque rang de page on peut définir un pied de page et une entête différents.

```
:default: &styleheader
:font: NomDeLaFont
:size: 13.5
:headers:
# Nom de l'entête, juste pour info, pour savoir ici ce que c'est
- :name:   "Nom de ce premier rang"
# Définition des pages qui utiliseront cet entête. Un rang de la
# première page à la dernière.
:pages: (12..15)
# Disposition de l'entête. Il est toujours constitué de 3 sections,
# le milieu, le côté gauche et le côté droit. Ils sont délimités
# par des "|". Le tiret '-' permet de définir l'alignement dans cette
# section :
#   -mot    => alignement à droite
#   mot-    => alignement à gauche
#   -mot-   => alignement au centre
:disposition: '%titre1- | | -%titre2'
# La police à utiliser. Elle doit impérativement être défini dans
# les :fonts:
:font: Arial
# Taille de la police (en points, je crois)
:size: 11
```



```
# Un autre rang
- :name:      'Nom de ce second rang' # juste pour information
  # etc.

#
# --- PIEDS DE PAGE ---
#
# Les définitions sont les mêmes que pour les entêtes.
:footers:
- :name: "Pied de page pour l'introduction"
  :pages: (1..5)
  :disposition:  ' | -%num- | '
  :font: Arial
  :size: 9
```

## Disposition

Le pied de page et l'entête sont divisés en trois parties de taille variable en fonction du contenu. Dans le format (`:disposition`), ces trois parties sont définies par des `|`.

L'**alignement** s'indique par des tirets avant, après ou de chaque côté du contenu. Quand le tiret est à droite (`mot-`), le mot est aligné à gauche, quand le tiret est à gauche (`-mot`) le contenu est aligné à droite, quand les tirets encadrent le contenu (`-mot-`) — ou quand il n'y en a pas — le contenu est centré.

## Variables

Les variables utilisables dans les entêtes et pieds de page sont toujours des mots simples commençant par `%`.

Pour les **niveaux de titre**, on utilise `%titre<NIVEAU>` par exemple `%titre4` pour les titres de niveau 4.

Pour les **numérotations**, on utilise `%num`. Noter que le contenu dépendra de la donnée `:num_page_style` de la recette du livre ou de la collection qui définit avec quoi il faut numéroté. TODO: à l'avenir on pourra imaginer avoir des numéros différents suivant les parties.

## Annexe

### Ne pas afficher les espaces insécables

Pour ne pas afficher les espaces insécables dans Sublime Text :

- Sublime Text > Préférences > Settings - Syntax specific
- ajouter dans la fenêtre droite :

```
{
  "draw_unicode_white_space": "none",
}
```

- enregistrer.

# Package Sublime Text

---

Pour travailler le texte, le mieux est d'utiliser un éditeur de texte. Sublime Text est mon éditeur de choix et on peut trouver dans le dossier `./resources/Sublime Text/` un package `Prawn4Book` qu'on peut ajouter au dossier `Packages` de son éditeur (dans Sublime Text, activer le menu "Sublime Text > Préférences > Browse packages..." et mettre le dossier `Prawn4Book` dans le dossier `Packages`).

L'application reconnaitra alors automatiquement les fichiers `.p4b.txt` et utilisera un aspect agréable, tout mettant en exergue les éléments textuels particuliers (comme les balises de formatage des paragraphes).

## Choix d'une autre police

Plus tard, la procédure pourra être automatisée, mais pour le moment, pour modifier la police utilisée dans le document `.p4b.txt` (ou markdown), il faut éditer le fichier `Prawn4Book.sublime-settings` du package et choisir la `"font_face"` qui convient (en ajouter une si nécessaire). Régler aussi le `"font_size"` et `"line_padding_top"` pour obtenir le meilleur effet voulu pour un travail confortable sur le texte.

## Prawn

---

### Blocs de texte avec Prawn