# Manuel de Praw-4-book



## LE PROGRAMME

## Introduction

***Prawn-for-book*** est le programme qui permet, à partir d'un texte formaté et d'une recette de produire un livre, c'est-à-dire un fichier PDF prêt à être envoyé à l'imprimeur pour une impression professionnelle.

> *Soyons tout de suite francs : après quelques petits réajustement inévitable, surtout si le livre est complexe.*

## La commande `pfb`

La commande **`pfb`** (pour « Prawn For Book ») permet d'accomplir toutes les tâches du programme. Mais pour le moment, il faut pour ça *l'installer*.

{TODO}

## Les trois opérations de base à connaitre

Pour lancer la construction du PDF :

```
$ pfb build
```

> *« build » signifie « construire », en anglais*

Pour ouvrir le fichier PDF :

```
$ pfb open
# Puis choisir "le livre PDF"
```

Pour ouvrir ce manuel :

```
$ pfb manuel
```

# LE LIVRE

## Introduction

Ce que nous appelons *livre* ici correspond au *fichier PDF* qui sera produit par le programme et sera envoyé à l'imprimeur, avec la couverture (produite avec d'autres outils) pour produire le livre publié.

## Créer un nouveau livre

La création d'un nouveau livre avec ***Prawn-for-book*** consiste à :

- créer le dossier du livre,
- initier le dossier à l'aide de la commande `pfb init`,
- définir le fichier recette initié,
- rédiger le fichier le texte initié,
- jouer la commande `pfb build` pour lancer la construction du livre avec le texte fourni en respectant la recette défini.

## Initier un nouveau livre

Ouvrir un Terminal dans le dossier du livre et jouer la commande :

```
$ pfb init
```

Suivez ensuite les opérations pour être accompagné(e) (ou non) dans le processus d'élaboration du livre.

## Initier une nouvelle collection

Ouvrir un Terminal dans le dossier de la collection, jouer la commande :

```
$ pfb init
```

Et choisir de faire une collection.

## Composer le livre

Rédiger le contenu du livre, appelé ici le TEXTE, définissez la RECETTE qui devra lui être appliquée.

Si nécessaire, programme des parseurs, des formateurs et des helpers qui vous permettront de gagner beaucoup de temps et d'obtenir un rendu une homogénéité à toute épreuve.

## Produire le PDF du livre

Jouer la commande `pfb build` pour produire le fichier PDF prêt pour l'impression (avec, notamment, les polices embarquées et les couleurs définies au format imprimerie).

### Les polices embarquées

Toutes les polices définies dans la recette sont embarquées dans le PDF final.

---

# LE TEXTE

Quand on dit « le texte » ici, on pense au « texte du livre à produire » et particulièrement le texte codé qu'il faut produire pour obtenir, en combinaison avec la recette le livre désiré.

## Les paragraphes

### Les paragraphes « stylés »

Il existe plusieurs moyens de styler des paragraphes particuliers :

- Par classe, à la manière du CSS en HTML. Cela permet d'uniformiser des mises en forme, sans avoir à les répéter.
- Par pré-paragraphe. Cela permet de définir de façon unique un paragraphe, en mettant ses informations dans la ligne qui le précède.

**Stylisation du paragraphe par classe**

La classe se met au début du paragraphe, avant des doubles deux-points « :: », en les séparant par des points s'il y a plusieurs classes :

```
class1.class2::Le paragraphe dans le style class1 et le style class2.
```

Il suffit ensuite de définir le style dans un fichier `formater.rb` à la racine du livre, dans un module `ParserFormaterClass`. La méthode à créer portera le nom de la classe, préfixé par `formate_` :

```ruby
# in formater.rb
module ParserFormaterClass

  # Définition simple (+str+ est le texte du
  # paragraphe)
  def formate_class1(str, context)
    return "<strong>#{str}</strong>"
    # => paragraphe en gras
  end

  # Définition pour utilisation complexe, en
  # se servant du "contexte", c'est-à-dire de
  # toutes les informations connues au moment
  # du traitement
  def formate_class2(str, context)
    par = context[:paragraph]
    par.font = "Arial"
    par.font_size = 14
    par.margin_left = "10%"
    par.kerning = 1.2
```

```
    par.margin_top = 4
    par.margin_bottom = 12
    par.text = "FIXED: #{par.text}"
    return par.text
  end
end
```

### Stylisation du paragraphe par pré-paragraphe

Pour une stylisation du paragraphe par « pré-paragraphe », on met dans la ligne précédente, entre `(( … ))` (comme tout code prawn-4-book) une table avec la définition des propriétés.

```
Un paragraph normal.

(( {align: :center, font_size: '16pt'} ))
Le paragraphe qui doit être aligné au centre et d'une taille de police de 16 points.
```

Noter que le code sera évalué tel quel ce qui signifie :

- on ne doit utiliser en clé (`align`, `font_size`, etc.) que des mots seuls ou des mots séparés d'un trait plat (contrairement à CSS on ne peut pas utiliser le tiret, dont font-size est mauvais) — voir ci-dessous la liste des propriétés possibles.
- on ne doit utiliser en clé que des valeurs évaluables donc : des nombres (`12`), des strings (`"string"`), des symboles (`:center`) ou des nombres avec unités connues (`12.mm` pour 12 millimètres). Ce qui signifie que `font_size: 12pt` est erroné tandis que `font_size: '12pt'` est correct.

Valeurs utilisables :

| PROPRIÉTÉ | DESCRIPTION | VALEURS |
|---|---|---|
| `font_family` / `font` | Nom de la fonte (qui doit exister dans le document) | String (chaine), par exemple `font_family:"Garamond"` |
| `font_size` / `size` | Taille de la police Entier ou valeurs. | `font_size:12`, `size: "12pt"` |
| `font_style` / `style` | Style de la police à utiliser (doit exister pour la police utilisée) | Symbol. `font_style: :italic` |
| `align` | Alignement du texte | :left, :center, :right, :justify |
| `kerning` | Espacement entre les lettres | Entier ou flottant. `kernel:2`, `kernel: "1mm"` |
| `word_space` | Espacement entre les mots | Entier ou flottant. `word_space: 1.6` |
| `margin_top` | Distance avec l'élément au-dessus | Entier en points-pdf ou valeur. `margin_top: 2.mm`, `margin_top: "2mm"` |
| `margin_right` | Distance avec la marge droite | idem |
| `margin_bottom` | Distance avec la marge du bas | idem |
| `margin_left` | Distance de la marge gauche | idem |
| `width` | Largeur de l'image (si c'est une image) ou largeur du texte. | Pourcentage ou valeur avec unité. `width: "100%"`, `width: 3.cm` |
| `height` | Pour une image, la hauteur qu'elle doit faire. | `height: "15mm"` |

### Les styles dans des paragraphes [mode expert]

Alors que ci-dessus nous avons vu comment styliser tout un paragraphe, dans sa globalité (ce qu'on appellerait un « style de paragraphe » dans un traitement de texte classique), ici nous allons voir comment mettre en forme du texte à l'intérieur du paragraphe (ce qu'on appellerait un « style de caractère » dans un traitement de texte).

Le plus simple est d'utiliser la fonctionnalité des index (en coulisse). On définit une méthode (pour éviter les problèmes de collision, l'essayer avant de l'utiliser). Voilà la démarche :

Choisir un nom de méthode, par exemple `ville`. L'essayer tout de suite avant de l'implémenter, pour éviter les collisions. Dans le fichier texte, écrire :

```
Je vis dans une ville qui s'appelle ville(Paris,75010).
```

Demander la fabrication du livre avec `$ pfb build`. Si ça produit une erreur, c'est parfait : la méthode n'existe pas. On peut l'implémenter, dans le ou un fichier `helpers.rb` à créer à la racine du dossier du livre.

```
module PrawnHelpersMethods
  def ville(params)
    return params
  end
end
```

Noter que même s'il y a deux paramètres dans `ville(Paris, 75010)`, ces deux paramètres arrivent en « Array » dans la méthode ci-dessus. Pour séparer, le nom de la ville de son code postal, on utilise toujours la même fonction :

```
module PrawnHelpersMethods
  def ville(params)
    nom, codep = params
    return nom
  end
end
```

Maintenant que tout est en place, on peut mettre en forme nos villes. Par exemple en la mettant en police `ArialN` (qu'on aura chargée dans la recette) et en italic.

```
module PrawnHelpersMethods
  TEMP_VILLE = '<em><font name"ArialN"> \
                      %{ville}</font></em>'
  def ville(params)
    nom, codep = params
    return TEMP_VILLE % {ville: nom}
  end
end
```

## Images

## Tables

## Sauts

Dans cette partie est abordé le problème des différents sauts, sauts de page principalement, pour se retrouver sur une page particulière (belle page ou page gauche) etc.

Pour insérer un saut de page, ajouter, seul sur une ligne, l'un des codes suivants :

```
(( new_page ))
ou
(( nouvelle_page ))
```

Pour insérer le nombre de sauts de pages nécessaires pour se retrouver sur une *belle page*, c'est-à-dire une page droite, ajouter, seul sur une ligne, l'un des codes suivants :

```
(( belle_page ))
ou
(( nouvelle_belle_page ))
ou
(( new_belle_page ))
ou
(( nouvelle_page_impaire ))
ou
(( new_odd_page ))
```

Enfin, pour insérer le nombre de sauts de pages nécessaires pour se retrouver sur une page gauche, ajouter, seul sur une ligne, l'un des codes suivants :

```
(( fausse_page ))
ou
(( new_even_page ))
ou
(( nouvelle_page_paire ))
```

## Commentaires

Les commentaires en ligne se marquent :

```
[#] Un commentaire sur une seule ligne.
```

Les blocs de commentaires se placent à l'aide de :

```
[#  Ceci est un bloc de
    commentaires qui tient
    sur plusieurs lignes
#]
```

On peut utiliser les marques de blocs de commentaire pour mettre du commentaire n'importe où dans le code. Par exemple :

```
(( {size:32 [# pour être assez grand #], align:right} ))
Ce paragraphe permet de montrer [# et bien montrer j'espère #]
qu'on peut mettre des commentaires à l'intérieur de ce qu'on
veut.
```

## Fichiers inclus

## Bibliographies

## Entête et pied de page

Les *entêtes* et *pieds de page* ne font pas à proprement parler partie du *texte du livre*. Ils sont traités en détail dans le RECETTE du livre.

## Formaters, parsers et helpers de texte

Sous ces trois noms qui peuvent vous sembler barbares se cachent un puissant moyen de simplifier la rédaction du livre et de lui donner un aspect parfaitement homogène quel que soit son (grand) nombre de page.

Cela donne également une grande souplesse à l'écriture, en permettant de modifier en profondeur un aspect, un affichage dans tout le livre en même temps et de le reporter même à plus tard.

Imaginons par exemple que vous ayez des noms de villes dans votre livre de voyage et que vous savez déjà que vous donnerez à ces noms de ville un aspect particulier à un moment ou à un autre, en tout cas pour la publication. Vous pouvez alors écrire votre texte de cette manière :

```
Ceci est mon texte qui parle de ville(Paris) mais aussi de
ville(Naples) ou de ville(Moscou).
```

Remarquez ci-dessus la balise `ville(...)`. Elle va permettre deux choses (au moins) : 1) de mettre en forme toutes les villes de la même manière et 2) de consigner toutes les villes citées dans le livre, en mémorisant même leur page et leur paragraphe. Pour la première utilisation, voir l'exemple de style dans un paragraphe.

---

# LA RECETTE

## Introduction

La *recette* est un fichier  'recipe.yaml' qui se trouve à la racine du dossier du livre ou de la collection. Elle définit le livre dans ses moindres détails et notamment comment devra être affiché le texte.

## Produire la recette

## Éléments de la recette

### Tailles du livre

### Polices chargées

### Pieds et entêtes de page

### Aspects des paragraphes

### Pagination

---

# ANNEXE

## Caractères spéciaux

Pour obtenir la liste complète des caractères spéciaux de la police « PictoPhil », jouer la commande `pfb pictophil`.

[RECETTE]: