Manuel de Prawn-for-book

(le manuel autoproduit de l'application de mise en forme professionnelle)

Philippe Perret

Icare Éditions

Avant-propos

Ce manuel présente toutes les fonctionnalités, à jour, de l'application « Prawn-for-book » (« Prawn pour les livres »), application dont la principale vocation est d'**obtenir un document PDF professionnel prêt pour l'imprimerie** à partir d'un simple fichier de texte (contenant le contenu du livre, le roman par exemple).
Ce manuel de XXX pages est auto-produit par « **Prawn-for-book** », c'est-à-dire qu'il est construit de façon ***programmatique*** par l'application elle-même. Il en utilise toutes les fonctionnalités puisqu'il génère de façon automatisé les exemples et notamment les ***helpers*** de mise en forme, les références croisées ou les bibliographies. En ce sens, ce manuel sert donc aussi de test complet de l'application puisqu'une fonctionnalité qui ne fonctionnerait pas ici ne fonctionnerait pas non plus dans le livre produit.
Si vous êtes intéressé(e) de voir comment il est généré, vous pouvez consulter principalement le fichier markdown `texte.pfb.md`, le fichier ruby `prawn4book.rb` et le fichier recette yaml `recipe.yaml` qui le définissent, dans le dossier `Manuel/manuel_building` de l'application.

Fonctionnalités

Stylisation par *class-tag*

Une autre manière de styliser du texte, qui demande plus de compétence mais pas forcément le expert_mode_expertniveau expert consiste à utiliser ce qu'on appelles les ***class-tag*** dans ***Prawn-For-Book*** ou les ***classes-balises*** en français.
Elle consiste à ranger en quelque sorte tous les attributs d'un paragrpahe sous un nom et d'utiliser ce nom devant le paragraphe pour lui affecter tous les attributs.
On peut voir cela comme les ***styles*** des ***feuilles de styles*** dans un traitement de texte classique.
« Classe », ici, est synonyme de « Style ».
On crée par exemple le ***style*** **citation** et on l'affecte au paragraphe contenant une citation :

```
Le paragraphe au-dessus de la citation.
citation::Rien ne sert de courir, il vaut mieux partir à temps.
Le paragraphe en dessous de la citation.
```

Le code ci-dessus pourra produire par exemple :
Le paragraphe au-dessus de la citation.

RIEN NE SERT DE COURIR,
IL FAUT PARTIR À TEMPS.

Le paragraphe en dessous de la citation.

Pour produire ce résultat, dans le fichier ruby `formater.rb`, j'ai implémenté le code suivant, qui ne demande pas une grande expertise malgré son apparence (en tout cas si vous n'avez jamais vu une ligne de code avant aujourd'hui) :

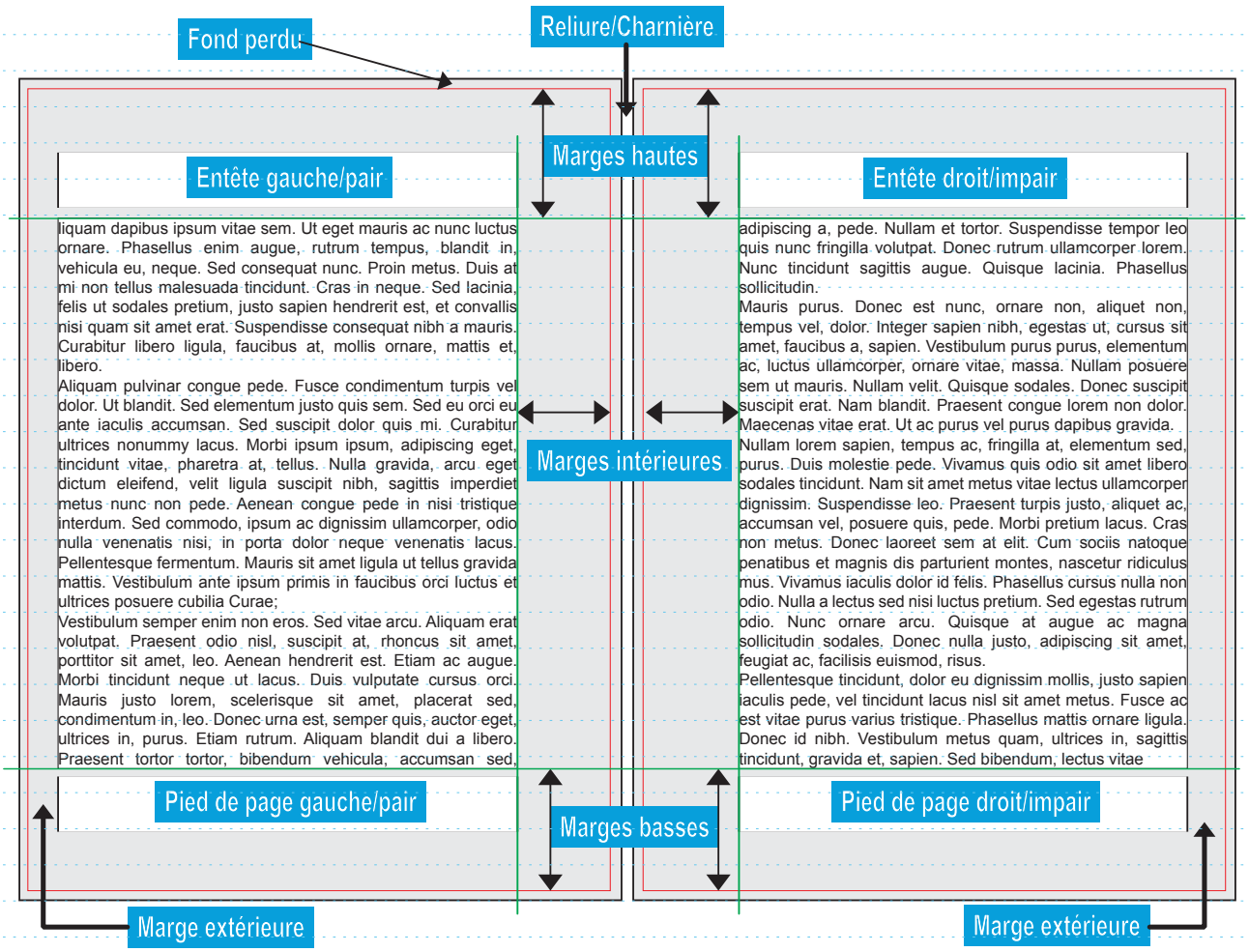
```
1   # Dans ./formater.rb
2
3   module ParserFormaterClass
4     def formate_citation\(str, context)
5       pr = context[:paragraph]
6       pr.align = :center
7       pr.lines_before = 4
8       pr.lines_after  = 4
9       pr.margin_left  = 6.cm
10      pr.margin_right = 6.cm
11      "<i>\#{str.upcase}</i>"
12    end
13  end
```

Expliquons les lignes de ce petit code...

Ligne 1, c'est un simple commentaire ruby, qui ne produit rien, il commence par un signe dièse et donne une indication. Ici, il nous dit juste qu'il faut mettre le code dans le (ou « un ») fichier de nom `formater.rb` qui doit se trouver à la racine de votre collection ou de votre livre.
Ligne 2, une ligne vide.
Ligne 3, on *ouvre* le module `ParserFormaterClass`. C'est un simple conteneur, on va mettre dedans toutes nos méthodes de *class-tags*.
Ligne 4, on définit la *méthode* (ou la *fonction*) qui va gérer notre *class-tag*. Puisque le *class-tag* s'appelle `citation`, notre méthode s'appelle obligatoirement `formate_citation`. On dit qu'on la définit grâce au `def` (pour *define*, « définir » en anglais) juste devant.
On trouve ensuite entre parenthèses les deux *paramètres* de cette méthode. Le premier est `str`, c'est en fait le texte qui suit la *class-tag* et les deux double-points dans notre texte. Dans notre exemple, `str` contient « Rien ne sert de courir, il faut partir à temps. ».
Noter que vous pouvez nommer ce paramètre comme vous voulez. Vous pouvez remplacer ce `str` par `texte` ou `barnabe` par exemple, c'est la même chose. La méthode a juste besoin de savoir comment s'appellera son premier paramètre.
Le second paramètre, `context`, est un peu plus compliqué. Nous l'expliqueront en détail dans le expert__titre_section_mode export, pour le moment, souvenez-vous simplement qu'il contient une propriété important, `:paragraph`, qui est le paragraphe contenant la citation. C'est lui que nous allons modifier pour obtenir l'aspect que nous voulons.
Tout ce qui suit l'ouverture de la méthode avec `def` concerne le traitement de notre texte, jusqu'au `end` de la ligne 12 (l'indentation nous permet de bien lire le code).
Ligne 5, nous récupérons le paragraphe dans le contexte et le mettons dans une variable appelée `pr`. C'est juste pour que ce soit plus simple à manipuler, pour ne pas avoir à répéter `context[:paragraph]` à toutes les lignes.
Ici aussi, nous l'avons appelé `pr` mais nous aurions pu choisir n'importe quoi d'autre, comme `par` ou `parag`.
Le seul nom qu'elle ne peut pas avoir, c'est le nom du premier paramètre de la méthode, dans lequel cas elle l'écraserait…
Maintenant que nous avons récupéré le paragraphe, nous allons le traiter pour l'apparence que nous désirons.
Ligne 6, nous indiquons que nous voulons que le paragraphe soit *centré*.
Ligne 7, nous indiquons que nous voulons laisser 4 lignes vides avant.
Ligne 8, nous indiquons que nous voulons 4 lignes vides après la citation.
Ligne 9 et **Ligne 10**, nous indiquons que nous voulons une marge à gauche et à droite de 6 centimètres.
Enfin, **ligne 11**, nous préparons le texte à renvoyer, celui qui sera écrit, en le formatant. Nous le mettons en capitales avec la méthode ruby `upcase` (`str.upcase` qui doit devenir `barnabe.upcase` si vous avez changé le nom du paramètre).
Et nous le mettons en italique grâce aux balises HTML « `<i>...</i>` ».
Nous avons fait un traitement de `str` un peu compliqué ici pour vous montrer ce qu'on peut faire, mais la plupart du temps, vous aurez juste à écrire `str` en fin de méthode pour que le texte soit retourné (n'oubliez pas de toujours le mettre, sinon c'est autre chose qui serait retourné).
Rappelez-vous bien : c'est toujours ce qui est à la fin de la méthode qui est retourné pour être marqué dans le document. Cette dernière ligne est donc très importante.
Ligne 12, nous *fermons* la méthode `formate_citation`.
Ligne 13, nous *fermons* le module `ParserFormaterClass`.
((line))
Maintenant, chaque fois que vous aurez une citation, vous pourrez la faire précéder de « `citation::` » et elle sera automatiquememnt formatée selon vos désirs. Comme ci-dessous.

Format du livre et des pages

Définition des marges



Des marges par défaut sont proposées, mais vous pouvez tout à fait définir celles que vous voulez très précisément, dans la recette du livre ou de la collection.
La seule chose à comprendre, par rapport aux documents dont vous avez l'habitude, c'est qu'ici les pages sont paires ou impaires, en vis-à-vis, et définissent donc :

- une marge haute et une marge basse (traditionnelle),
- une marge *intérieure*, qui comme son nom l'indique est à l'intérieur du livre, près de la reliure, de la *charnière*, du *dos du livre* (souvent confondu avec la *tranche du livre*),
- une marge *extérieure*, qui comme son nom l'indique est tournée vers l'extérieur du livre, vers la tranche (la vraie cette fois).

Ces marges sont donc définies par les propriétés `top` (« haut » en anglais) `bot` (pour « bottom », « bas » en anglais), `ext` (pour « extérieur ») et `int` (pour « intérieur »).

NB : Quels que soient les réglages, il y aura toujours un ***fond perdu*** (« bleeding » en anglais) de 10 pps (points-postscript) autour de la page. C'est la « marge » que s'accorde l'imprimeur pour découper le livre.
Traditionnellement, la marge intérieure est plus large que la marge extérieure, car une bonne partie de cette marge est prise dans la reliure.
De la même manière, la marge basse est plus large que la marge haute car elle contient le numéro de page. Il peut cependant arriver que la marge haute contienne un entête.
Pour régler parfaitement les marges, vous pouvez soit ajouter l'option `-margins` à la commande `pfb build` qui construit le livre, soit mettre le `show_margins` de la recette à true, comme nous l'avons fait ci-dessous.
Dans l'exemple ci-dessous nous avons volontaire ***pousser*** les valeurs pour rendre bien visibles les changements.

SI la recette du livre contient...

```
book_format:
  page:
    margins:
      top: 40mm
      bot: 20mm
      ext: 2mm
      int: 35mm
    show_margins: true
```

ET que le texte du livre contient...

Pour cette page, où les marges sont visibles, on illustre des marges de page à 40 mm en haut, 20 mm en bas, 35 mm à l'intérieur et 2 mm à l'extérieur.
Vous remarquez donc une immense marge en haut, une grande marge en bas, une marge externe toute petite (*ce qui ne serait pas du tout bon pour une impression de livre*) et une marge intérieure moyenne.
Remarquez aussi que le texte est automatiquement justifié, il s'aligne parfaitement sur le bord de ces marges gauche et droite, ce qui donne un rendu impeccable.

ALORS le livre contiendra...

Les marges sont visibles grâce au `show_margins` mis à `true` dans la recette. On aurait pu aussi jouer la commande avec l'option `-margins`.

Pour cette page, où les marges sont visibles, on illustre des marges de page à 40 mm en haut, 20 mm en bas, 35 mm à l'intérieur et 2 mm à l'extérieur.
Vous remarquez donc une immense marge en haut, une grande marge en bas, une marge externe toute petite (*ce qui ne serait pas du tout bon pour une impression de livre*) et une marge intérieure moyenne.

1

Remarquez aussi que le texte est automatiquement justifié, il s'aligne parfaitement sur le bord de ces marges gauche et droite, ce qui donne un rendu impeccable.

2

Multi-colonnage

On peut provisoirement passer en double colonnes ou plus grâce à la marque :

```
(( colonnes(<nombre de colonnes>) ))
```

… où `<nombre de colonnes>` est, comme on le devine, à remplacer par le nombre de colonnes que l'ont veut obtenir.
C'est ce qu'on appelle le *mode multicolonnage*.
Pour quitter le mode multi colonnes, il suffit de revenir à une seule colonne :

```
(( colonnes(1) ))
```

Grâce à cette marque, on peut avoir autant de colonnes que l'on désire, comme ci-dessous.
Un paragraphe au-dessus de la section multi-colonnes.
Ce texte se trouve en mode multi-colonnes, avec trois colonnes, grâce à la marque `(( colonnes(3) ))` qui se trouve au-dessus et la marque `(( colonnes(1) ))` qui se trouve en dessous. Il ne faut pas oublier la dernière marque.
Un paragraphe au-dessous de la section multi-colonnes.

Précaution pour les sections multi-colonnes

Attention à toujours terminer par `(( colonnes(1) ))`, même lorsque c'est la dernière page. Dans le cas contraire, en cas d'oubli, la section multi-colonnes — et tout son texte — ne sera tout simplement pas gravée dans votre livre.

Définition plus précise des colonnes

Comme pour tout élément ***Prawn-For-Book***, le comportement par défaut est harmonieux et devrait apporter satisfaction à tout utilisateur. Mais comme pour tout élément, on peut néanmoins redéfinir de façon précise de nombreux élément, simplement en ajoutant un deuxième paramètre à la méthode `colonnes` après le nombre de colonnes.
Ce paramètre est une table ruby, donc entre accolades, avec des paires « `propriété: valeur` ».
Par exemple :

```
(( colonnes\(\<nombre cols>, {\<prop>: \<valeur>, \
     \<prop>: \<valeur>, etc.}) ))
```

Largeur de gouttière entre les colonnes

On appelle *gouttière* l'espace vertical laissé entre deux colonnes. On peut le redéfinir avec la propriété `gutter` (« gouttière » en anglais).

```
(( colonnes(2, {gutter: 40}) ))
```

Un paragraphe normal situé au-dessus de la section à double colonnes.

Une section en multi-colonnes (2) avec une gouttière de 4 centimètres. Une gouttière aussi large sépare très bien les deux colonnes. On peut utiliser cette fonctionnalité aussi pour placer une image entre les deux colonnes.

Un paragraphe normal situé sous la section à double colonnes.

Largeur totale occupée par les colonnes

On peut définir aussi sur quelle largeur les colonnes devront tenir, par exemple la moitié de la page :

```
(( colonnes(2, width: PAGE_WIDTH/2) )) [1]
```

[1] *Vous remarquez ci-dessus l'utilisation d'une constante (cf. annexe_constantesannexe/constantes).*

Lignes avant et après la section multicolonnes

Grâce à `lines_before` et `lines_after`, on peut définir le nombre de lignes à laisser entre le texte et la section en multi-colonnes. Il s'agit très précisément du *nombre de lignes vides.* avant (`lines_before` — "lignes avant" en anglais) et après (`lines_after` — "lignes après" en anglais) la section multi-colonnes.
Par exemple, avec :

```
(( colonnes(2, {lines_before: 2, lines_after:3} ))
```

... on laissera 2 lignes vides entre le paragraphe précédent et le début de la section à 2 colonnes, et 3 lignes vides entre la fin de la section multi-colonnes et le paragraphe suivant.

Distance exacte avant et après la section multi-colonnes

On peut utiliser de la même manière `space_before` et `space_after`, en leur donnant comme valeur une distance (en points-postscript, en millimètre, etc.) mais les propriétés `lines_before` et `lines_after` ci-dessus doivent être préférées pour conserver un aspect impeccable par rapport à la annexe_grille_referenceannexe/grille_reference.

Styles du texte dans les multi-colonnes

On peut modifier les paragraphes de façon générale à l'intérieur d'une section multi-colonne grâce au deuxième paramètre.
Les propriétés modifiables sont :

– **`align`** | Alignement des paragraphes. Justifiés (`:justify`) par défaut, on peut les aligner à gauche (`:left` ou `LEFT`) ou à droite (`:right` ou `RIGHT`). La section ci-dessous est obtenue avec le code
`(( colonnes(2, {align: RIGHT}) ))`

Un paragraphe normal situé au-dessus de la section à double colonnes.

Ce texte est aligné à droite dans la section double colonne grâce à la propriété `align` mise à `RIGHT`. Rappel : par défaut, le texte est justifié.

Un paragraphe normal situé sous la section à double colonnes.

– **`font`** | Fonte utilisée pour le texte. C'est une annexe_font_stringannexe/font_string classique. La section ci-dessous est engendrée par le code
`(( colonnes(2, {font:"Arial/bold/8.5/008800"}) ))`.

Un paragraphe normal situé au-dessus de la section à double colonnes.

Une section en double colonnes avec la police « Arial », le style « bold », une taille de 8.5 et une couleur vert foncé (008800). Par défaut, c'est la police du livre qui est utilisée.

Un paragraphe normal situé sous la section à double colonnes.

Très long texte en multi-colonnes

Si vous avez un extrêmement long texte en multi-colonnes (par exemple tout votre livre), il est préférable de le diviser en plusieurs sections plutôt que de le faire tenir dans une seule section (ce qui entrainement fatalement des problèmes).
Par exemple, renouvelez la marque `(( colonnes(2) ))` à chaque nouveau chapitre (sans oublier de terminer la section précédente par `(( colonnes(1) ))`.

Ajouter ou retrancher des lignes en multi-colonnes

Malgré tous nos efforts, ou pour des besoins propres, il est possible que les colonnes ne correspondent pas à ce que l'on attend au niveau de leur hauteur.
Dans ce cas, grâce à la propriété `add_lines`, on peut ajouter ou retrancher un certain nombre de lignes. Par exemple, ci-dessous, on force l'affichage dans une seule colonne à l'aide de `(( colonnes(2, {add_lines: 3}) )`.

Un paragraphe normal situé au-dessus de la section à double colonnes.

Un texte qui devrait tenir sur les deux colonnes mais qu'on a allongé en hauteur pour avoir 3 lignes de plus. Donc ce texte tient intégralement sur la première colonne comme voulu par le nombre.

Un paragraphe normal situé sous la section à double colonnes.

Nombre fixe de lignes en multi-colonnes

De la même manière et malgré tous nos efforts, il est possible de fixer de façon précise le nombre de lignes.
On peut utiliser soit la propriété `lines_count` pour définir le nombre de lignes (c'est la valeur) que doit avoir la section multi-colonnes, soit la propriété `height` pour définir la hauteur avec un unité de mesure (pouces, millimètre, etc.).

```
                (( colonnes(3, {lines_count: 10}) ))
# ...
(( colonnes(3, {height: "4cm"}) ))
```

Un paragraphe normal situé au-dessus de la section à double colonnes.

Cette section multi-colonnes est obtenue en ajoutant la propriété `lines_count` à 5. On voit que cette colonne contient bien cinq lignes texte et qu'ensuite seulement on passe dans la colonne suivante pour écrire le reste du texte.

Un paragraphe normal situé sous la section à double colonnes.

Styles des paragraphes dans les sections multi-colonnes

On peut tout à fait utiliser la texte_detail_inline_stylingtexte_detail/inline_styling pour styliser les paragraphes dans une section multi-colonnes.

Un paragraphe normal situé au-dessus de la section à double colonnes.

Un premier paragraphe normal dans la section multi-colonnes.
Ce paragraphe rouge avec de l'*italique* et du **gras** est aligné à droite.

Paragraph centré.
Paragraph aligné à gauche en Helvetica light de 9 en bleu.
Un paragraphe indenté.

Un paragraphe normal situé sous la section à double colonnes.

Pour le moment, seules les propriétés `lines_before`, `lines_after`, `margin_left` et `margin_right` ne sont pas prises en compte. Pour les premières, il suffit d'ajouter des lignes vides avec le caractère « espace insécable » ([ALT] [ESPACE]).

Icare Éditions

Conception
Philippe Perret
(philippe.perret@icare-editions.fr)

Mise en page
Prawn-For-Book

Couverture
MM & PP

Correction & relecture
Marion Michel et Philippe Perret

Imprimé par Prawn-For-Book