Prawn For Book Manuel

Prawn For Book
Manuel Introduction Présentation Les grandes forces de Prawn-for-book Commande principale Obtenir de l'aide AIDE RAPIDE **BIBLIOGRAPHIE IMAGES** TABLES (TABLEAU) **HEADERS/FOOTERS** Créer un livre Création d'une collection Ajouter un livre à une collection Construction du PDF du livre Ouvrir le fichier PDF produit Options de fabrication (travail du livre) Ouverture du PDF Rédaction du texte du livre Package Sublime Text Le livre pour l'impression Les marges **Pagination** Les titres Grand titre sur une belle page Exclure un titre de la table des matières Les paragraphes Définition Les différents types de paragraphe Évaluation du texte du paragraphe (interprétation des variables) PARAGRAPHES DE TEXTE **TITRES IMAGES Images SVG** Numéro de paragraphe pour l'image **TABLES** Insertion d'une table Définition précise de la table Attributs des cellules Valeurs en pourcentage Valeurs implicites Protection du texte

```
Fusion de cellules
        Quelques exemples concrets
        Définir un STYLE DE TABLE
        BLOC DE CODE
        Suppression de la numérotation
        Suppression de l'espace autour des tables
        Réglage de l'espace autour de la table
        Table pour style de paragraphe [Expert]
    Les paragraphes-codes (pfb-code)
    Numérotation des paragraphes
Commentaires dans le texte
Passer une ligne vierge
Saut de page
    Saut de page dans le programme [Expert]
Insertion d'un texte externe
Production d'une page dynamique (ou de plusieurs) (Expert)
HEADERS & FOOTERS (entêtes et pieds de page)
    Principe
    Rangs de pages
    Contenu
    Définition des entêtes et pieds de page
    Positionnement
    Tiers et contenus
    Dans la recette
    Variables
Pages spéciales
    Page de titre
    Page d'informations
    Table des matières
    Page d'index
    BIBLIOGRAPHIES
        La balise de la bibliographie
        Le titre de la bibliographie
        Chemin d'accès aux données de la bibliographie
        La page de la bibliographie
        Les données de la bibliographie
        Mise en forme des données bibliographiques
        Mise en forme de l'item de bibliographie dans le texte
RÉFÉRENCES (et références croisées)
    Références croisées
Exclure des paragraphes
Commentaires
PARSE(U)RS & FORMATE(U)RS
    Méthode d'erreurs (gestion des erreurs)
Message de notification
    Méthode d'helpers
    Formatage personnalisé (formater.rb)
        Formatage des paragraphes
```

Insérer une image dans une cellule

```
Tailles de police proportionnelles
            Formatage des éléments de bibliographie
            Formatage des tables
        Parsing personnalisé des paragraphes (parser.rb)
RECETTE DU LIVRE
    Présentation générale
        Création de la recette du livre
        Contenu de la recette du livre
    Éléments de la recette
        book_data (informations générales du livre)
        collection_data (données pour la collection)
        book_format (format du livre)
        titles (données d'affichage des titres)
        publishing (données de la maison d'édition)
        fonts (données des polices)
            Dossiers des fontes
        bibliographies (données bibliographiques)
        table_of_content (données de table des matières)
        inserted_pages (types de page à imprimer)
            Impression forcée des pages de type
        page_de_titre (définition de la page de titre)
        page_info (définition de la page des informations)
        page_index (données d"affichage de la page d'index)
Modules personnalisés (Expert)
Les scripts [experts]
    Lancer un script
    Créer un script
        Arguments passés au script
Annexe
    Sous-commandes
    Exemples de codes
        Exemple de méthode générale par paragraphe code
    Style inline
    Snippets
        Snippets personnalisés
    Conversion des dimensions
    Grille de référence
    Points PDF
Ne pas afficher les espaces insécables
Trouver le chemin vers une police
Package Sublime Text
    Choix d'une autre police
    Modifier l'aspect du texte dans Sublime Text (son affichage dans l'application)
Prawn
    Blocs de texte avec Prawn
```

Insertion d'un titre par un helper

Introduction

Présentation

Prawn4book — ou **Prawn For Book**, c'est-à-dire « Prawn pour les livres » — est une application en ligne de commande permettant de transformer un simple texte pseudo-markdown en véritable PDF prêt pour l'imprimerie, grâce au (formidable) gem **Prawn** qui donne son nom à l'application.

Sa commande (qui doit être installée) est : pfb (« Prawn For Book »).

L'application met en forme le texte, dans ses moindres détails et ses moindres aspects, empaquette les polices nécessaires, gère les références — même les références croisées —, gère les index et les bibliographies — autant que l'on veut —, gère les mises en place complexes pour produire un PDF conforme en tous points aux désirs de l'auteur ou de l'éditeur.

Les grandes forces de Prawn-for-book

Les grandes forces de *PRAWN-FOR-BOOK* sont donc :

- production simple d'un document PDF valide, professionnel, prêt pour l'imprimerie,
- gestion cohérente de toute une collection de livres,
- mise en forme du texte dans ses moindres détails (feuilles de style, modules complexes experts de formatage),
- gestion des références internes (renvois, références à une page ou un paragraphes, etc.),
- gestion des références croisées (références à la page d'un autre livre)
- gestion d'un index,
- gestion d'autant de bibliographies que l'on veut,
- gestion automatiquement de la table des matières (est-ce vraiment utile de le préciser?...),
- gestion puissante de n'importe quelle information au fil du texte ou sous forme de rapport,
- extension infinie des capacités... (pour les experts)

Commande principale

Sa commande simple est (*):

\$> pfb

Ou en version longue (*):

\$> prawn-for-book

(*) En présupposant bien sûr que des alias de commande ont été créé, sur MacOs grâce à :

 ${\tt ln -s} \ / {\tt Users/me/Programmes/Prawn4book/prawn4book.rb} \ / {\tt usr/local/bin/prawn-for-book}$

ln -s /Users/me/Programmes/Prawn4book/prawn4book.rb /usr/local/bin/pfb

Et sur Windows grâce à:

Pour les sous-commandes de Prawn-for-book, voir l'annexe.

Obtenir de l'aide

On peut obtenir de l'aide de différents moyens :

- \$> pfb aide ouvrir une aide générale en présentant les commandes principales.
- \$> pfb aide de definit de l'aide sur l'<identifiant>. On peut obtenir grâce à cette commande les assistants de création qui permettent de définir très précisément la recette d'un livre ou d'une collection.
- \$> pfb lexique "groupe de mots" offrira de l'aide sur un mot particulier ou un groupe de mots en en donnant la définition ou le sens dans *Prawn-for-book*. Note : les guillemets ne sont nécessaires que s'il y a plusieurs mots.
- Pour ouvrir le manuel : \$> pfb manuel (ajouter -dev pour l'ouvrir en édition).

•

AIDE RAPIDE

BIBLIOGRAPHIE

Voir comment utiliser une bibliographie.

IMAGES

Voir comment insérer une image dans le texte.

TABLES (TABLEAU)

Voir comment insérer une table ou un tableau dans le texte.

HEADERS/FOOTERS

Voir comment <u>définir les pieds et page et entêtes de page</u>.

Créer un livre

Créer un livre avec **Prawn-for-book** consiste à créer deux choses [1], deux fichiers:

- le <u>fichier recette</u> recipe.yaml qui définit tous les aspects du livre, en dehors du contenu textuel luimême,
- le <u>fichier texte</u> texte.pfb.md qui contient le texte du livre.

[1]

En réalité, il suffit même d'un seul fichier : le fichier texte avec le nom exact texte.pfb.md, et toutes les autres valeurs seront définies par défaut.

Pour créer ces deux éléments de façon assistée, suivez simplement cette procédure :

- Choisir le dossier dans lequel doit être créé le livre,
- ouvrir une fenêtre Terminal dans ce dossier,
- jouer la commande \$> pfb init
- choisir de construire un nouveau livre,
- suivre l'assistant pour définir les données du livre (ou n'en définissez aucune, vous aurez toujours le loisir de le faire plus tard).

Création d'une collection

Avec **Prawn-for-book**, on peut aussi créer des collections, c'est-à-dire un ensemble de livres qui partageront les mêmes éléments, à commencer par la charte graphique. Plutôt que d'avoir à la copier-coller les informations de livre en livre (informations de mise en forme, de collaborateurs, d'édition, etc.), entrainant des opérations lourdes à chaque changement, on crée une collection qui définira les éléments communs et on met les livres dedans.

Pour créer une collection :

- Choisir le dossier dans lequel doit être créée la collection,
- ouvrir une fenêtre Terminal à ce dossier,
- jouer la commande \$> pfb init
- choisir de construire une collection,
- suivre l'assistant de création.

Tous les livres de la collection devront se trouver dans le dossier de la collection, à la racine.

dossier_collection	
dossier_livre_1	
dossier_livre_2	
recipe_collection.yaml	
dossier_ressources	

Ajouter un livre à une collection

Suivre la <u>procédure d'initiation d'un nouveau livre</u> mais en ouvrant le Terminal au dossier de la collection (ou au dossier du livre créé dans le dossier de cette collection).

Construction du PDF du livre

Pour produire le fichier PDF qui servira à l'impression du livre, jouer la commande \$> pfb build dans le dossier du livre :

- > cd path/to/book/folder
- > pfb build

À bien noter : cette commande fabrique vraiment le PDF qu'il suffira d'envoyer à l'imprimeur pour tirer le livre.

Ouvrir le fichier PDF produit

Pour ouvrir le document PDF à la fin de la fabrication, ajouter l'option —open à la commande build.

\$> pfb build -open

Options de fabrication (travail du livre)

Certaines options de la commande build permettent de définir les attributs du livre plus facilement (les marges, la grille de référence, etc.).

Résultat de l"option	Option
Affichage des marges sur chaque page	-display_margins
Pour limiter à un nombre de pages	-grid=X-Y -display_margins - grid=X-Y
Affichage de la g <u>rille de référence</u>	-display_grid
Production du livre jusqu'à la page X	-last=X
Affichage de la position du cursor vertical (sa hauteur sera ajoutée en début de paragraphe).	-c/-cursor

Ouverture du PDF

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

Note: on doit se trouver dans le dossier du livre.

Rédaction du texte du livre

On peut travailler le texte du livre dans n'importe quel éditeur simple. <u>Sublime Text</u> 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, **avec le plugin PrawnForBook**, que le texte se termine par **.pfb.txt** ou **.pfb.md** pour que Sublime Text applique le format *Prawn4Book*.

Package Sublime Text

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

\$> pfb open package-st

Voir en annexe comment modifier l'aspect du texte dans Sublime Text

Le livre pour l'impression

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 :



basse	
Fond perdu	_
-	_

Ce qui signifie que le haut et le bas du texte sont calculés en fonction des marges et des header (entête) et footer (pied de page).

Noter qu'il y a toujours un fond perdu de 10 post-script points autour de la page. Je crois qu'on ne peut pas le modifier.

Pagination

Recette	propriété	valeurs possibles
book_format:page:	:numerotation	pages/parags

Une des grandes fonctionnalités de *Prawn-for-book* est de permettre de paginer de deux manières :

- à l'aide des numéros de pages (pagination traditionnelle),
- à l'aide des numéros de paragraphes (pagination "technique" permettant de faire référence à un paragraphe précis, par son numéro/indice).

La numérotation des paragraphes peut être très pratique aussi quand on veut recevoir des commentaires précis — et localisés — sur son roman ou tout autre livre. Vous pouvez l'utiliser pour le PDF que vous remettez à vos lecteurs et lectrices.

Pour se faire, on règle la valeur de la propriété **book_format:page:numerotation** dans la <u>recette du livre</u> <u>ou de la collection</u>. Les deux valeurs possibles sont <u>pages</u> (numérotation des pages) ou <u>parags</u> (<u>numérotation des paragraphes</u>).

Modifier la valeur directement dans le fichier recette du livre ou de la collection nécessite une certaine habitude. Il est préférable, pour tous les réglages, de passer par les assistants. Ici, il suffit de jouer \$> pfb assistant et de choisir "Assistant format du livre", puis de renseigner la propriété "Numérotation".

Cette valeur influence de nombreux éléments du livre, dont :

- les numéros en bas de page (si on les désire)
- les <u>index</u>
- les repères bibliographiques
- les marques de <u>références</u>

Pour savoir comment placer et formater les numéros de pages, cf. <u>Headers et Footers</u>.

Les titres

La base du texte étant du markdown, les titres s'indiquent avec des dièses en fonction de leur niveau :

```
# Grand titre
## Titre de chapitre
### Titre de sous-chapitre
#### Titre de section
etc. si nécessaire.
```

Pour la mise en forme des titres dans le livre, voir la définition des titres dans la recette du livre.

Grand titre sur une belle page

Pour qu'un grand titre se retrouve toujours sur une belle page (ie la page impaire, à gauche), on doit mettre sa propriété :belle_page à true dans la recette du livre ou de la collection.

Pour la mise en forme des titres dans le livre, voir les titres dans la recette du livre.

Exclure un titre de la table des matières

Pour exclure un titre de la table des matières, c'est-à-dire pour qu'il soit inscrit en tant que titre dans le texte mais qu'il n'apparaissent pas dans la table des matières, il suffit de mettre {no-tdm} dans ce titre, n'importe où sauf avant les dièses. Par exemple :

```
# {no-tdm} Titre exclus de la tdm
# Titre dans la tdm
## Autre titre exclus {no-tdm}
```

Les paragraphes

Définition

L'unité textuel de *Prawn-for-book* est le paragraphe (mais ce n'est pas l'atome puisqu'on peut introduire des éléments sémantiques dans le paragraphe lui-même, qui seront évalués "en ligne").

Les différents types de paragraphe

- les <u>Paragraphes de texte</u>,
- les <u>Titres</u>,
- les <u>Images</u>,
- les Pfb-codes.

Évaluation du texte du paragraphe (interprétation des variables)

Un texte de paragraphe est souvent constitué de variables qui dépendent du contexte ou sont définis pour un livre. En règle générale, ces variables sont estimées au cours de la fabrication du livre.

Mais parfois, il est nécessaire de forcer cette interprétation des variables. C'est le cas, par exemple, pour le Paradigme de Field Augmenté fabriqué dans les analyses de film des éditions Icare.

Dans ce cas, il faut explicitement appeler la méthode Prawn4book::PdfBook::AnyParagraph.__parse en lui fournissant les bons arguments:

```
str_corriged = Prawn4book::PdfBook::AnyParagraph.__parse(<string>, <context>)

# avec <context> qui doit absolument définir :

# :pdf [Prawn::PrawnView] Le document en cours de fabrication

# :paragraph [NTextParagraph|PFBCode] Instance du paragraphe contenant le texte
```

Rappel: pour obtenir :pdf, se souvenir que plusieurs méthodes de parsing et de formatage personnalisées peuvent utiliser leur nombre de paramètres pour déterminer les informations qui seront transmises.

PARAGRAPHES DE TEXTE

Le paragraphe de texte se définit simplement en l'écrivant dans le fichier .pfb.md.

```
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.
```

Il existe ensuite plusieurs manières de styliser ces paragraphes si nécessaire :

- stylisation par défaut,
- stylisation en ligne de portion de textes dans le paragraphe,
- stylisation inline (en ligne),
- stylisation d'un extrait par helper,
- <u>stylisation par balise initiale</u>.

STYLE PAR DÉFAUT DU PARAGRAPHE

Recette	propriété	valeurs possibles	
	:default_font_n_style:	Nom de fonte (police) chargée et le style utilisé	"Garamond/italic"
	:default_font_size:	Nombre entier ou flottant	12.4
	:default_font_style	[OBSOLÈTE] Un des styles défini pour la fonte	

On définit le style du paragraphe par défaut dans la <u>recette du livre ou de la collection</u> en définissant les propriétés :default_font (nom de la fonte, qui <u>doit être chargé dans le document</u>), :default_font_size (taille de la police) et :default_font_style (style défini pour la fonte, en général ':nomal'.

STYLE DE PORTIONS DE TEXTES DANS LE PARAGRAPHE

Le paragraphe peut contenir de la mise en forme simple, "en ligne", comme le gras ou l'italique, en entourant les mots avec <i>...</i> ou Par exemple :

```
Un mot en <b>gras</b> et un mot en <i>italique</i>. Une expression en <i><b>gras et
italique</b></i>.
<!-- Ou, en pseudo-markdown -->
Un mot en **gras** et un mot en *italique*. Une expression __soulignée__. Noter les
deux traits plats de chaque côté.
```

STYLISATION "INLINE" DU PARAGRAPHE — (({<hash}))

Un paragraphe peut être complètement modifié en utilisant ce qu'on appelle la *stylisation inline* qui consiste à ajouter une ligne juste au-dessus du paragraphe qui contient ses propriétés modifiées. Par exemple :

```
Un paragraphe au style par défaut.

(( {<data>} ))
Le paragraphe influencé par les <data> ci-dessus.
```

Noter les ((...)) (doubles-parenthèses) qui sont la marque de Prawn-for-book et les crochets qui vont définir une table de propriété (un *dictionnaire*, comme dans un langage de programmation.

On peut, à la base, changer par exemple la taille du texte pour ce paragraphe avec la propriété :font size.

```
(( {font_size:22} ))
Ce paragraphe aura une taille de 22 pour la police courante.
```

La propriété font family permet de changer de fonte (à nouveau il faut que cette fonte soit accessible).

```
(( {font_family: "Arial"} ))
Ce paragraphe sera en Arial, dans la taille par défaut de la police par défaut.
```

On peut mettre plusieurs propriétés en les séparant par des virgules :

```
(( {margin_left: 40, margin_top: 50} ))
IMAGE[images/mon_image.svg]
```

L'image ci-dessus se retrouvera à 40 <u>points-pdf</u> de la marge gauche et à 50 <u>points-pdf</u> de son contenu précédent.

Les propriétés qu'on peut définir sont les suivantes :

Propriété	Description	Valeurs
font_family	Nom de la fonte (qui doit exister dans le document)	String (chaine), par exemple <pre>font_famiily:"Garamond"</pre>
font_size	Taille de la police	Entier ou valeurs. P.e. font_size:12
font_style	Style de la police à utiliser (doit être défini pour la police)	Symbol (mot commençant par ":"). P.e.: font_style: :italic
kerning	Éloignement des lettres	Entier ou flottant. P.e. kernel:2
word_space	Espacement entre les mots	Entier ou flottant. P.e. word_space: 1.6
margin_top	Distance avec l'élément au-dessus	Entier en points-pdf ou valeur. P.e. margin-top: 2.mm
margin_right	Distance avec la marge droite	Idem
margin_bottom	Distance avec l'élément inférieur	Idem
margin_left	Distance de la marge gauche	Idem
width	Largeur de l'image (si c'est une image)	Pourcentage ou valeur avec unité. P.e. width: "100%" ou width: 3.cm (notez qu'il n'y pas de guillemets lorsqu'on utilise les unités Prawn.
height	Pour une image, la hauteur qu'elle doit faire.	

AJUSTEMENT DU PARAGRAPHE

Une propriété particulièrement utile pour de l'impression professionnelle concerne l'espacement entre les mots qui permet d'éviter les mots seuls en fin de paragraphes par exemple. Supprimer deux ou trois mots sur la dernière ligne peut permettre par exemple de faire remonter un titre de façon élégante.

Pour gérer cette fonctionnalité, on utilise la commande ((del_last_line)) ("delete the last line", "supprimer la dernière ligne"). L'application joue alors elle-même sur l'espacement entre les mots (voire entre les lettres) pour condenser un peu le texte.

Par mesure de prudence, pour obtenir un rendu acceptable, n'appliquez jamais cette commande s'il y a trop de mots sur la ligne à supprimer et/ou si le paragraphe est trop cours. Un paragraphe de moins de 4 lignes se met en danger si on lui applique cette commande.

Exemple d'utilisation:

```
(( del_last_line ))
Ceci est un texte assez long qui doit être condensé
pour que sa dernière ligne soit supprimée, en jouant
sur les espacements entre chaque mot, en les rapprochant
de façon discrète pour que les trois derniers mots soient
rayés de la carte.
```

Le paragraphe pouvant avoir plusieurs définitions, on peut utiliser aussi la commande comme propriété :

```
(( {del_last_line:true, font_size:10.2} ))
Ceci est un texte assez long qui doit être condensé
...
```

Après traitement, le paragraphe ressemblera à :

```
Ceci est un texte assez long qui doit être condensé pour que sa dernière ligne soit supprimée, en jouant sur les espace-ments entre chaque mot, en les rapprochant de façon discrète pour que les trois derniers mots soient rayés de la carte.
```

Bien entendu, cette commande ne se place dans le texte du livre que lorsque le PDF a été construit et qu'on a constaté l'état du paragraphe. On ne peut pas le faire au hasard, il faut le faire comme le ferait un metteur en page, sur pièce.

STYLISATION D'UN EXTRAIT DU PARAGRAPHE PAR HELPER [Expert]

[Expert] On peut créer une méthode ruby pour mettre en forme (ou tout autre chose) en la définissant en helper. Par exemple, si je veux mettre dans une forme spéciale des horloges, je peux utiliser :

```
Je suis arrivé à #{time('0:12:25')} et je repartirai à #{time('0,30,0')}.
```

Pour ce faire, on implémente dans le fichier helpers.rb de la collection ou du livre :

```
# in ./helpers.rb
module Prawn4book
class PdfBook::NTextParagraph # ou AnyParagraph
  def time(str)
    # ... traitement de +str+...
    return str
  end
end
end
```

Attention aux collisions de nom, le nom de la méthode utilisée ne doit pas exister dans le programme.

TIP : pour s'en assurer, il suffit d'appeler la méthode dans le texte avant de l'implémenter. Si une erreur est produite, informant que la méthode n'existe pas, alors c'est bon.

STYLISATION DU PARAGRAPHE PAR BALISE INITIALE

Un paragraphe de texte peut également commencer par une *balise*, qu'on appelle ici **class-tag**, qui va déterminer son apparence, son *style* comme dans une feuille de styles. Ces balises peuvent être <u>communes</u> (<u>propres à l'application</u>) ou <u>personnalisées</u>.

Personnalisation des paragraphes texte (style de paragraphe personnalisés) [Expert]

Les *styles de paragraphes personnalisés* doivent être identifiés par une *balise* qui sera placée au début du paragraphe à stylisé. Par exemple, si ma balise est gros, cela donnera :

```
gros::Le paragraphe qui sera mis dans le style personnalisé "gros".
```

Ensuite, pour fonctionner, il faut dire à *Prawn-for-book* comment styliser ce paragraphe.

Il existe deux manières de le faire :

• la manière simple, en ne se servant que des propriétés ci-dessus. Dans cette utilisation, le style permet simplement de ne pas avoir à répéter toute la ligne de définition du paragraphe avant le paragraphe.

Pour cette manière, il faut définir dans le module ParserFormaterClass du fichier formater.rb la méthode formate_<class_tag>(str, context) qui reçoit en premier paramètre le texte à traiter et en second paramètre son contexte. Ensuite, à l'intérieur de cette méthode, on définit toutes les valeurs :

```
module ParserFormaterClass
 # Utilisation simple
 def formate_gras(str, context)
   return "<strong>#{str}</strong>"
 # Utilisation complexe
 def formate_gros(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
```

• la manière complexe, permettant une gestion extrêmement fine de l'affichage, mais nécessitant une connaissance précise de Prawn.

Noter qu'on peut aussi <u>utiliser la puissance des tables</u> pour mettre en forme de façon très précise les paragraphes.

La manière complexe consiste à définir dans le module FormaterParagraphModule du fichier formater.rb la méthode buildparagraph(paragraph, pdf) qui reçoit en premier argument l'instance du paragraphe et en second argument l'instance Prawn::View du constructeur du livre. Ensuite, à l'intérieur de la méthode, on construit le paragraphe. Par exemple :

```
module FormaterParagraphModule

def build_gros_paragraph(par, pdf)

pdf.update do
   font(par.font, size: par.font_size)

bounding_box([100, cursor], width: bounds.width/2, height: 100) do
    transparent(0.5) { stroke_bounds }
   image icone_tip, at: [...]
   text par.text,
   end
   end
end
end
```

Styles paragraphes texte commun

Balise	Description	Exemples
dict::entry:: [TODO]	Entrée de dictionnaire	
dict::text:: [TODO]	Description de l'entrée, le texte suivant l'entrée.	

TITRES

Le titre se définit comme en markdown c'est-à-dire à l'aide de dièses.

```
# Un grand titre
## Un chapitre
### Un sous-chapitre
etc.
```

Pour voir comment insérer un titre dans un helper, cf. <u>Insertion d'un titre par un helper</u>.

IMAGES

Les images se définissent à l'aide de la balise :

```
IMAGE[<data>]
```

Les données sont composées d'un chemin d'accès à l'image, puis de données qui définissent l'image. Le **chemin d'accès** doit être soit absolu soit relatif.

Tip: Il est préférence de mettre les images dans un dossier images se trouvant dans le dossier du livre ou de la collection et d'y faire référence simplement par images/mon_image.jpg.

Les images peuvent être de tout format, mais puisqu'elles sont destinées à l'impression, leur espace colorimétrique doit être le <u>modèle colorimétrique CMJN (Cyan, Magenta, Jaune, Noir</u>).

```
Ci-dessous une image qui sera présentée sur toute la largeur de la page (hors-marge).

IMAGE[images/pour_voir.jpg|width:100%]

L'image gardera de l'air avant ce texte, même s'il est collé dans le texte.

Une image qui sera réduite de moitié.

IMAGE[images/red.jpg|width:50%]
```

Images SVG

Pour une raison qui m'échappe pour le moment, lorsque l'on utilise une image syg produite avec *Affinity Publisher*, même lorsque l'on ne prend que la partie conservée, l'image occupe une place plus grande, presque une image.

Il faut utiliser **inkscape** pour *rogner* l'image en ses bords naturels. Pour procéder à cette opération :

- ouvrir un Terminal dans le dossier contenant l'image
- jouer la commande :

```
> inkscape -l -D -o image-rogned.svg image.svg
```

• => l'image sera rognée, c'est celle-ci qu'il faut utiliser dans le livre.

Numéro de paragraphe pour l'image

Par défaut (pour le moment), les images ne sont pas numéroter comme des paragraphes (seuls les paragraphes de texte le sont). Pour numéroter une image, il suffit cependant de laisser un paragraphe avant qui ne contient qu'une espace insécable.

Il faut vraiment que ce soit une insécable, sinon le paragraphe sera passé.

Cela ne fonctionne pas non plus si on utilise ((line))

Propriétés de l'image

Trouvez ci-dessous la liste des propriétés qui peuvent être utilisées pour les images :

Propriété	Description	Valeurs possibles
width	Dimension de l'image par rapport à elle-même	Pourcentage, valeurs fixes
width_space	Quantité d'espace horizontal que l'image doit couvrir, en pourcentage. 100% signifie que l'image doit couvrir toute la largeur de la page même les marges.	Pourcentage
TODO		

TABLES

Les tables sont certainement le meilleur moyen de formater des paragraphes de façon particulière sans trop de complexité/difficulté. Par exemple, si l'on désire une boite de cadre avec un fond de couleur particulière, utiliser une table se révèle beaucoup plus pratique et flexible que toute autre solution qui utiliserait les propriétés des bounding_box (es) de Prawn, par exemple.

Notez que dans ce cas, une table se réduit très souvent à définir une classe de table et mettre le texte entre traits droits, de cette manière :

```
(( {table_class: :ma_table} ))
| Le texte qui doit se présenter d'une manière particulière |
```

Insertion d'une table

On peut insérer une table dans le code à l'aide du formatage classique de l'extension de markdown:

```
| Titre 1 | Titre 2 | Titre 3 |
| :--- | :---: | ---: |
[ Colonne 1 | Colonne 2 | Colonne 3 |
etc.
```

Note: au niveau du traitement, on n'utilise pas *Kramdown*, qui sortirait un code HTML alors que **Prawn** ne gère pas le formatage HTML. En fait on utilise le gem **Prawn-table**.

Ci-dessous, on remarque qu'une entête est définie (ligne de données avant les ____) et que l'alignement de chaque colonne est défini. Ce sont les mêmes alignements qu'en markdown, mais avec un nouveau : [_____] (noter qu'aucune espace n'est laissée avant et après les []) qui signifie qu'il faut justifier le texte dans la colonne.

Définition précise de la table

On peut définir très précisément la table avec un ligne de code avant, défini entre crochets comme c'est l'usage avec **Prawn-for-book**. Par exemple :

```
Un paragraphe de texte normal.

(( {column_widths: [100,100, 20]} ))
| Large | Large | Petite |
| Content | Content |
```

Attributs des cellules

```
:column_widths
                 Largeur de chaque colonne.
                 Array, largeur de chaque colonne.
                 Unité : PS-points ou pourcentage
                 On peut aussi ne définir la dimension que de certaines colonnes, en
                 donnant en valeur une table qui contient en clé l'indice 1-start de
                 la colonne est en valeur la dimension. Par exemple :
                 { column widths: {2 => '20%'} }
:width
                 Largeur de la table (par défaut adaptée au contenu - mais il vaut
mieux
                 la spécifier explicitement) [1]
:header
                 Si true, la première rangée est considérée comme une entête.
                 Pour positionner la table. Valeur
:position
                 :left (positionner à gauche), :right (positionner à droite) :center
                 (positionner au centre) XXX (positionner à xxx ps-points.
                 [<even color>, <odd color>] pour mettre alternativement les deux
:row colors
                 couleurs à chaque rangée. <even color> et <odd color> sont des
                 hexadécimaux (par exemple "F0F0F0").
                 [Hash] Pour définir le style des cellules. Les paramètres sont :
:cell style
                 * :width (largeur de cellule),
                 * :height (hauteur de cellule),
                 * :padding Padding de la cellule, soit un nombre soit [top, right,
                   bottom, left])
                 * :padding top, :padding right, :padding bottom, :padding left
                 * :borders => [<liste des bords à mettre>] (p.e. [:left, :top]
                 * :border_width => xxxx Épaisseur du trait
                 * :border color Couleur du bord
                 * :border_lines => Le style de lignes. Soit une valeur seule, parmi
                   :solid, :dotted ou :dashed soit un Array de 4 valeurs pour définir
                   dans l'ordre : ligne haut, droit, bas et gauche.
                              La fonte à utiliser
                 * :font style Le style
                 * :size
                              La taille de police
                 * :align
                              L'alignement, parmi les valeurs traditionnelles
                               (:justify, :right, :left)
                 * :text_color Couleur de texte (hexadécimale)
```

[1]

Afin de ne pas être confronté au problème de taille qui ne correspond pas au contenu dans les tables, il est bon de préciser explicitement la largeur de la table attendu. On peut s'appuyer évidemment sur pdf.bounds.width qui contient la largeur de page utilisable (donc la largeur totale de la page à laquelle a été retiré les marges).

Valeurs en pourcentage

Par défaut, *Prawn-table* ne connait que les valeurs fixes. On peut cependant fournir des valeurs en pourcentages, qui seront traitées en fonction de la taille.

Rappel: on peut utiliser pdf.bounds.width pour obtenir la largeur utilisable de la page.

Valeurs implicites

Certaines valeurs peuvent être données de façon implicite. Le cas le plus classique concerne les colonnes. Si une colonne doit avoir une largeur adaptable en fonction des deux autres, on ne définit pas sa largeur. On utilise alors la valeur nil. Par exemple :

```
Une table avec largeur de colonne implicite.

(( {width: "50%", col_count:3, column_widths: [10, nil, 10]} ))
| A1 | B1 | C1 |
| A2 | B2 | C2 |
```

Noter la définition de :col_count ici qui est importante car c'est cette valeur qui va permettre de savoir, en cas de colspan ou de rowspan combien il y a réellement de colonnes (dans le cas où il y aurait <nombre de colonnes> - 1 éléments dans la liste :column_widths, avec donc une dernière colonne qui ne serait pas précisée explicitement.

Note : si la valeur :width (largeur de la table) n'est pas définie, alors c'est par défaut une table qui prendra toute la largeur de la table.

Protection du texte

Différents caractères sont « dangereux » pour les tables, à commencer par le trait droit (« | ») qui permet de délimiter en pseudo-markdown les tables. Pour obtenir un texte sans problème, quand on insert du texte dans une table, on peut utiliser la méthode de classe Prawn4book::PdfBook::NTable::safeize:

```
safe_str = Prawn4book::PdfBook::NTable.safeize(bad_str)
```

Insérer une image dans une cellule

Pour insérer une image dans une cellule, utiliser **IMAGE[path|style]** où path est le chemin absolu ou relatif de l'image et style est optionnellement le style à appliquer à l'image. Par exemple :

```
Ci-dessous un table qui contient une image.

| La belle image | IMAGE[images/mon_image.jpg|scale:0.5] |
```

Les attributs des styles peuvent être :

On peut aussi utiliser toutes les définitions attributs des cellules.

Fusion de cellules

Pour fusionner des cellules, on utilise **colspan** et **rowspan** comme en HTML. Mais dans ce cas, il faut définir la cellule avec une table (Hash) dont la propriété :content définira le contenu textuel.

Par exemple:

```
Ci-dessous une table avec des cellules fusionnées.

| A | B | C |
| {content: "A+B", colspan: 2} | C |
| {content: "3+4" rowspan: 3} | B | C |
| B | C |
| B | C |
```

Noter que si les <u>valeurs implicites</u> sont utilisées et que des colspans ou rowspan aussi, il est extrêmement prudent de définir aussi <u>:col_count</u> pour définir explicitement le nombre de colonnes dans la table, afin que les calculs des valeurs implicites puissent se faire correctement.

Quelques exemples concrets

Une table sans aucun bord:

```
(( {cell_style:{border_width: 0}} ))
| A1 | B1 | C1 |
| A2 | B2 | C2 |
```

Une table avec des bords horizontaux

```
(( {cell_style:{border_width: [1,0]}} ))
| A1 | B1 | C1 |
| A2 | B2 | C2 |
```

Une table avec des bords verticaux

```
(( {cell_style:{border_width: [0, 0.5]}} ))
| A1 | B1 | C1 |
| A2 | B2 | C2 |
```

Définir un STYLE DE TABLE

Si plusieurs tables sont similaires, plutôt que d'avoir à remettre pour chacune tous les attributs, on peut définir un style de table avec la propriété :table_class . Au-dessus de la table, il suffira d'indiquer :

```
(( {table_class: :ma_table_customisee} ))
| Valeur | valeur |
...
```

Attention, **NE PAS OUBLIER LES ":"** avant le nom du style.

Noter qu'on peut aussi définir d'autres paramètres que la classe, même quand celle-ci est définie.

Ensuite, dans <u>le fichier formater.rb</u> on doit définir une méthode au nom du style de table (ici ma_table_customisee qui va recevoir l'instance PdfBook::NTable et retournera les options à ajouter à la construction de la table. Ces options sont les propriétés définissables ci-dessus.

Par exemple:

```
# Dans formater.rb

module TableFormaterModule

def table_ma_table_customisee(ntable)
    # ... Traitement peut-être des lines ...
    # En modifiant @lines
    return {column_widths: [100,50,50]}
    end
end
```

On peut par exemple ajouter une image seulement dans cette méthode plutôt que d'avoir à la mettre dans toutes les tables. Par exemple, pour les exemples du SRPS avec un smiley souriant et un smiley grimace, on peut imaginer de faire ceci :

Dans le texte:

```
Ceci est un paragraphe quelconque.

(( {table_class: :smiley_sourire} ))

| | C'est bien de faire comme ça |

Un autre paragraphe quelconque.

Et puis un autre.

(( {table_class: :smiley_grimace} ))

| | Ça n'est pas bien de faire comme ça |

| | Ça n'est pas bien non plus comme ça |

Un autre paragraphe encore.
```

Et dans le fichier parser.rb, on place:

```
# in formateur.rb
module TableFormaterModule
 def table_smiley_sourire(ntable)
   ntable.lines.each do |line|
      line[0] = {image: smiley path(:sourire)}
   end
   return smiley_style
  end
 def table smiley grimace(ntable)
   ntable.lines.each do |line|
      line[0] = {image: smiley_path(:grimace)}
   end
   return smiley_style
  end
 def smiley_style
    @smiley style | = {column widths: [50, 200]}
 end
 def smiley_path(which)
   return File.join(IMAGE FOLDER, "smiley #{which}.jpg")
 end
end
```

BLOC DE CODE

Mais on peut utiliser aussi la méthode de classe pour définir un bloc de code qui va utiliser la possibilité de bloc sur la table pour un usage très puissant qui permet de cibler exactement ce que l'on veut. Se rapporter au manuel pour voir toutes les possibilités.

On le fait de cette manière :

```
<!-- dans le texte -->

(( {table_class: :avec_bloc} ))

| A1 | B1 | C1 |

| A2 | B2 | C2 |
```

Et dans formater.rb :

```
module TableFormaterModule

def table_avec_bloc(ntable)
  ntable.code_block = Proc.new do
  # ... ici le traitement ...
  # par exemple :
    cells.width = 50
    cells.align = [:center, :left, :right]
    # Voir dans le manuel Prawn-Table toutes les possibilités end
end
end
```

Par exemple:

```
module TableFormaterModule

def table_avec_bloc(ntable)

#

# Modification du contenu d'une cellule

# (première cellule de la 3e rangée)

#

ntable.lines[2][0] = "000"

#

# Ajout d'une ligne à la table :

#

ntable.lines << ["D1","D2","D3"]

#

# Remplacement d'une rangée complète

#

ntable.lines[2] = [{content: "Oh !", colspan: 3}]</pre>
```

```
# @note
      On peut utiliser "blockcode", "code_block" ou "block_code"
  ntable.blockcode = Proc.new do
     # column(0) concerne la première colonne
    # columns(0) Idem
                 Ces deux valeurs agissent sur toutes les cellules
                 de la première colonnes
    cs = column(0)
    cs.font = 'Arial'
    cs.size = 24
    cs.width = 100 # largeur de la colonne
    cs.font_style = :bold
    cs.background_color = 'DDDDDD'
    cs.text color = 'CC0000'
    return nil # IMPORTANT
  end
  end
end
```

Note 1

Grâce à ces possibilités, on peut faire une utilisation très puissante des tables, avec par exemple des données injectées depuis des fichiers de données externes. Voir par exemple l'utilisation avec les paradigmes de Field augmentés.

Note 2

On ne peut pas régler les colspan et rowspan dans le bloc de code. Si on doit le faire au niveau de la définition de la table, il faut le faire en travaillant sur les lignes, comme cela est expliqué ici

On peut utiliser le bloc de code pour filtrer des cellules :

```
module TableFormaterModule

def table_filtrante(ntable)

#

# Utilisation du filtre "naturel" avec les méthodes

# :select

#

cells.filter do |cell|
    cell.content.match?("oui")
    end.background_color = '00FF00'

#

# Utilisation d'un bloc de filtrage ruby
```

```
#
bons = Prawn::Table::Cells.new
bads = Prawn::Table::Cells.new

columns(1..-2).rows(2..12).each do |cell|
    next unless cell.content.numeric?
    if cell.content.to_i > 25
        bons << cell
    else
        bads << cell
    end
end

bons.background_color = "00CC00"
bads.background_color = "FF0000"

end
end</pre>
```

Note : la classe Prawn::Table::Cells est un Array qui possède des méthodes supplémentaires et permet notamment de boucler sur chaque cellule qu'il contient.

On peut également récupérer n'importe quelle valeur passée dans le "pfbcode" (le code entre doublesparenthèses avant la table) en invoquant :parag_style. Par exemple :

```
Ceci est une table :

(( {table_class: :matable, variable1: "Aujourd'hui"} ))
| A1 | A2 | A3 |
| B1 | B2 | B3 |
```

Et dans le formater :

```
module TableFormaterModule

def matable(ntable)

#
    # Noter qu'on l'efface en la prenant, pour éviter les problèmes

#
    var1 = ntable.parag_style.delete(:variable1)

matable.blockcode = Proc.new do
    #
    # On met cette valeur dans la première ligne de la première colonne
    #
    column(0).row(0).content = var1
    end
```

```
#
# On pourrait aussi faire ça pour changer le contenu de la cellule
#
ntable.lines[0][0] = var1
end
end
```

Suppression de la numérotation

Quand la numérotation par paragraphe est activée, on peut supprimer la numérotation des tables en utilisant :

```
table.print(pdf, **{numerotation: false})
```

Suppression de l'espace autour des tables

Par défaut, un espace est laissé entre les tables. On peut le supprimer complètement en utilisant :

```
table.print(pdf, **{no_space: true})
```

Réglage de l'espace autour de la table

Par défaut, l'espace est bien calculé autour d'une table. On peut cependant le régler précisément avec :

```
table.print(pdf, **{space_before: 2.mm, space_after: 1.02689.cm})
```

Noter la façon d'écrire les distances, en profitant des méthodes de Prawn::Measurements.

Table pour style de paragraphe [Expert]

Il est possible d'utiliser la puissance des tables pour mettre en forme un style de paragraphe sans passer par les tables (c'est-à-dire sans passer par la définition (({table_class: ...})). C'est particulièrement utile pour la mise en forme des bibliographies et autres index par exemple.

Le principe général est de créer une instance Prawn4book::PdfBook::NTable et de l'imprimer.

Une instance Prawn4book::PdfBook::NTable reçoit deux arguments: le pdfbook et les données de la table:

```
table = Prawn4book::PdfBook::NTable.new(pdfbook, table_data)
```

Les données table_data de la table contiennent deux données, une pour définir les lignes, l'autre pour définir le style, en passant par un PFBCode :

```
table_data = {
  pfbcode: <instance Prawn4book::PdfBook::PFBCode>,
  lines: liste Arraydes lignes>
}
```

L'instance Prawn4book::PdfBook::PFBCode doit être créée en fournissant le pdfbook (qu'on peut obtenir de pdf par pdf.pdfbook) et la ligne string définissant la table. Par exemple :

```
pfbcode = Prawn4book::PdfBook::PFBCode.new(pdf.pdfbook, "(( {col_count:3, column_widths: 30} ))")
```

TIP 1 : Plutôt que de faire une liste de lignes (difficile à gérer si on utilise des valeurs template), on peut utiliser cette tournure :

TIP 2: De la même manière, plutôt que d'avoir à écrire en string le PFBCode qui va définir la table, on peut utiliser la tournure suivante :

```
pfbcode = {
col_count: 3,
column_widths: [20, 100],
cell_style: {
  padding: 0
}
}
pfbcode_string = "(( #{pfbcode.inspect} ))"
```

Noter que pour fonctionner, il faut que la méthode reçoive pdf, comme c'est le cas des mises en forme de bibliographie.

Voici un exemple pour mettre en forme un style de paragraphe par balise (tag) :

Dans le texte, on trouve :

```
synopsis::Le synopsis de la scène.
```

Ce texte va donc appeler la méthode ParserFormaterClass#formate_synopsis qui devrait retourner le texte à écrire. Mais nous voulons mettre le synopsis en forme de façon particulière et les tables sont bienvenues.

Nous utilisons donc:

```
# in ./parser.rb
module ParserFormaterClass
 def formate_synopsis(str, context)
   pdf = context[:pdf]
   pdfbook = pdf.pdfbook
   # Nous demandons l'écriture du numéro de paragraph car le
   # livre fonctionne en numérotant les paragraphes
   context[:paragraph].print paragraph number(pdf)
   # On protège le texte à insérer dans la table (il ne doit
   # pas, entre autres choses, contenir de trait droit.
   str = Prawn4book::PdfBook::NTable.safeize(str)
    # Puis nous définissons la table
   data_table = {
      pfbcode: Prawn4book::PdfBook::PFBCode.new(pdfbook, "(( #{aspect.inspect} ))")
      lines: ["| | #{str} |"]
    table = Prawn4book::PdfBook::NTable.new(pdfbook, **data table)
   # Nous imprimons le synopsis, tout près du texte précédent
   table.print(pdf, **{numerotation:false, no space: true})
  end
 # @return les données Prawn pour la table
 def aspect
    {
     col_count: 2,
      column widths: [80, nil]
      cell style: {
       padding: 0,
       border width: 0
        }
      }.freeze
  end
end
```

Voici un exemple pour afficher de façon complexe un item de bibliographie :

```
# in ./formater.rb
module BibliographyFormaterModule # définition pour les bibliographies
 # Cette méthode sert à formater les items de la bibliographie 'livre' dans
 # l'affichage de fin de livre
 def biblio livres(bibitem, pdf) # ces deux arguments sont toujours fournis
   pdfbook = pdf.pdfbook
   # Aspect de la table
   data = {
      col_count: 2,
     width: 100,
      column widths: [20, nil],
      cell style: {
       padding: 0
     }
    }
    # Définition de la table
   table data = {
      pfbcode: Prawn4book::PdfBook::PFBCode.new(pdfbook, "(( #{data.inspect} ))"),
      lines: [
        "| %{title} | %{auteur} | " % bibitem.temp data,
                   | %{resume} | " % bibitem.temp_data, # [1]
      ]
    }
    # Instanciation de la table
   table = Prawn4book::PdfBook::NTable.new(pdfbook, table data)
   # Impression de la table
   table.print(pdf, **{numerotation: false, no_space: true})
   # Il faut retourner nil pour que le texte ne soit pas écrit dans le pdf
   # puisque cette méthode s'en charge elle-même avec `table.print(pdf)'
   return nil
  end
end
```

[1] Pour cette utilisation, **penser à supprimer tous les retours chariots** des textes insérés en utilisant par exemple :

```
texte = texte.strip.gsub(/\r?\n/,'<br />')
```

Dans le cas contraire, les tables seraient tronquées.

Code Prawn

Si on est expert de Prawn, on peut aussi passer par le code direct.

```
# pfd [Prawn::PrawnView]

pdf.update do
  table(lines, aspect) do |tb|
   tb.row(0).style(borders:[:top, :left, :right], border_width: 1})
  end
end
```

Avec:

```
aspect = {
  width: taille,
  column_widths: largeur des colonnes
  cell_style: ...
  # etc. même données que ci-dessus
}

lines = [
  ["Première colonne première ligne", "Deuxième colonne première ligne"],
  ["first column deuxième ligne", "2e colonne 2e ligne"],
  # etc.
]
```

Les paragraphes-codes (pfb-code)

Ces paragraphes sont des paragraphes simples, contenant un seul "mot-programme", et permettent notamment de gérer le contenu du livre. Ce ne sont donc pas à proprement parler des paragraphes de texte mais ils auront une influence réelle sur le livre produit. On trouve par exemple :

```
Pour passer la suite à la page suivante :

(( new_page ))

Pour l'inscription de l'index :

(( index ))

Pour l'inscription de la table des matières :

(( tdm ))

Pour l'inscription d'une bibliographie :
```

```
(( biblio(films) ))
Etc.
```

Paragraphes-codes de méthodes personnalisées

Ces paragraphes-codes peuvent aussi définir du code personnalisé. Ils peuvent alors être de trois sortes :

- le paragraphe code qui retourne du texte à écrire,

 Utiliser le module ParserFormaterClass dans parser.rb pour le définir.
- le paragraphe code qui écrit un texte dans le livre,

 Utiliser le module ParserFormaterClass dans parser.rb pour le définir.
- le paragraphe code qui ne produit aucun texte mais reçoit une donnée utile pour la suite.

 Utiliser le module Prawn4book dans prawn4book.rb pour le définir en tant que méthode de classe.

```
# in ./prawn4book.rb (collection ou livre)
module Prawn4book
def self.<methode>(<paramètres>) # [1]
  # ... traitement ...
end
end
```

[1] Les paramètres de la méthode vont déterminer ce qui doit être envoyé à la méthode. S'il n'y a aucun argument, la méthode sera appelée sans arguments, même si des arguments sont passés au code par erreur. Si le nombre d'arguments attendus correspond au nombre d'arguments fourni, ils sont transmis tels quels. Si le nombre de paramètres de la méthode est supérieur au nombre d'arguments fournis, le pdf est lui aussi transmis à la méthode. Cela permet alors d'imprimer quelque chose dans le texte.

Voir les exemples d'appel à une méthode générale.

Numérotation des paragraphes

	Recette	propriété	valeurs possibles
		:numerotation:	pages (défaut), parags
		:num_parag:	Table de valeurs
Ajustement de hauteur		parag_num_vadjust	Integer
Distance avec le texte		:parag_num_dist_from_text	Integer
Taille de la police		:parag_num_size	Integer
Force (opacité)		:parag_num_length	>0 - 100

Pour un livre technique, où les références sont fréquentes, ou si l'on veut que l'index ou les bibliographies renvoient à des endroits très précis du livre, il peut être intéressant de numéroter les paragraphes. Pour ce faire, on met la propriété :parags de la recette du livre ou de la collection à true.

```
book_format:
  book:
   numerotation: pages # ou parags
  text:
   numerotation: pages # ou parags
```

L'affichage utilise par défaut la police <code>Bangla</code>, mais elle peut être définie grâce à la propriété <code>:num_parag</code> de la recette, après s'être assuré que cette fonte était définie dans les <u>fontes</u> du livre ou de la collection :

Le chiffre peut ne pas être tout à fait ajusté au paragraphe. Dans ce cas, on utilise la propriété
parag_num_vadjust pour l'aligner parfaitement. La valeur doit être donnée en pixels PDF, elle doit être assez faible (attention de ne pas décaler tous les numéros vers un paragraphe suivant ou précédent.

```
book_format:
# ...
text:
# ...
numerotation: parags
parag_num_vadjust: 1
parag_num_dist_from_text: 12
parag_num_size: 9
parag_num_strength: 72
```

Noter ci-dessus qu'on peut également demander à ce que <u>la numérotation des pages</u> se fasse sur la base des paragraphes et non pas des pages (pour une recherche encore plus rapide).

Non numérotation des tables

Si les <u>tables sont utilisées pour la mise en forme des bibliographies</u>, on peut demander de ne pas mettre la numération en utilisant :

```
table.print(pdf, **{numerotation: false})
```

Commentaires dans le texte

On peut insérer des commentaires dans le texte à l'aide du code <!-- ... --> (le même que celui utilisé en HTML).

Mais à la différence du HTML, pour le moment, on ne doit utiliser cette balise que sur une ligne seule, pas au bout d'un texte :

```
Un paragraphe de texte.
<!-- Ce commentaire est valide --> 
Un paragraphe de texte.<!-- Commentaire invalide -->
```

Note : les émoticones ne doivent bien sûr pas être utilisés de cette manière, ils ne sont là que pour commenter l'utilisation .

Passer une ligne vierge

Ajouter à l'endroit voulu :

```
(( line ))
```

Noter que cette ligne ne sera pas numérotée.

Saut de page

Prawn-for-book gère automatiquement les passages à la page suivante lorsque le texte arrive en bas de page. On peut cependant tout à fait forcer un saut de page pour forcer le passage à la page suivante à l'endroit voulu. On utilise dans le texte, **seul sur un paragraphe**, l"une de ces deux marques :

```
(( new_page ))
<!-- OU -->
(( nouvelle_page ))
```

Notez la forme d'une *commande Prawn-for-book* (elles permettent d'affiner l'impression du livre jusque dans le moindre détail) :

- la double parenthèse
- l'espace laissée de chaque côté de cette parenthèse, entre la commande et la parenthèse intérieure.

Si l'on veut se retrouver **sur une page paire** (donc une page de gauche), utiliser l'une de ces marques :

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

Si l'on veut se retrouver sur une **page impaire** (donc la page de droite), utiliser l'une de ces marques :

```
(( new_odd_page ))
ou
(( nouvelle_page_impaire ))
ou
(( new_belle_page ))
```

Saut de page dans le programme [Expert]

Dans le code ruby, donc pour tout ce relève des *formaters*, on ajoute un saut de page en faisant pdf.start_new_page

Insertion d'un texte externe

On peut insérer un autre fichier pfb.md (ou autre...) dans le texte texte.pfb.md d'un livre Prawn. Pour ce faire, il suffit d'utiliser la commande include: suivie du chemin relatif ou absolu du fichier.

Par exemple, si le dossier du livre contient un dossier textes et un fichier texte introduction.pfb.md contenant le texte de l'introduction, on peut l'insérer dans le livre à l'endroit voulu à l'aide de :

```
(( include: textes/introduction ))
```

Noter que ci-dessus aucune extension de fichier n'a été nécessaire. Elle n'est utile que s'il existe plusieurs fichiers de même affixe (nom sans l'extension) dans le dossier. Dans le cas contraire, **Prawn-for-book** recherche le fichier dont il est question.

Production d'une page dynamique (ou de plusieurs) (Expert)

[Expert] Parfois, on peut vouloir produire une page dynamiquement, avec les données récoltées en cours de fabrication du livre. C'est le cas par exemple pour les analyses de film où un scénier est produit automatiquement à la fin des livres en relevant les intitulés au cours de la fabrication.

On utilise dans ce cas, dans le texte, à l'endroit où l'on veut que la page soit imprimée, la tournure :

```
(( ma_methode_de_fabrication_de_la_page ))
```

... et l'on définit cette méthode dans helpers.rb:

```
module PrawnHelpersMethods

def ma_methode_de_fabrication_de_la_page

pdf.update do
    text "Mon texte pour la page".
    end
end
end
```

Noter que la méthode a accès à pdf (Prawn::View) pour imprimer dans le livre et pdfbook pour obtenir toutes les informations sur le livre.

Des arguments peuvent être transmis à cette méthode :

```
<!-- dans le texte markdown -->

Je vais afficher ici les tarifs des abonnements.

(( affichage_tarifs(:abonnement) ))

Vous savez à quoi vous en tenir.
```

Et dans le module PrawnHelpersMethods :

```
module PrawnHelpersMethods

def affichage_tarifs(what)
   case what
   when :livre then ...
   when :abonnements then ...
   end
   end
end
```

HEADERS & FOOTERS (entêtes et pieds de page)

Par défaut (c'est-à-dire sans aucune précision), seul le pied de page est construit, avec le numéro de la page au milieu. Mais il est possible de définir finement chaque entête (header) et chaque pied de page (footer) et même d'en créer autant que l'on veut, tout à fait différents, pour les différentes sections du livre.

Principe

Chaque **pied de page** (*footer*) et chaque **entête** (*header*) est une partie contenant trois sections appelées des "**TIERS**", un à gauche, un à droite et un au milieu de chaque page gauche et droite, où sont définis les éléments à afficher.

Pour gérer les entêtes et pieds de page, on crée des DISPOSITIONS qui comprennent les données suivantes :

- un nom humain pour mémoire,
- un rang de pages sur lequel appliquer la disposition,
- un headfooter pour l'entête des pages gauche et droite (cf. ci-dessous),
- un headfooter pour le pied des pages gauche et droite,
- une valeur d'ajustement vertical du pied de page et de l'entête,
- un identifiant.

On crée autant de dispositions que nécessaire.

Rangs de pages

Une disposition est définie pour un rang de pages qui peut être défini explicitement grâce aux paramètres :first_page et :last_page.

Contenu

Le contenu de chaque TIERS, quelconque, peut être :

- le numéro de la page,
- le numéro du paragraphe,
- le nom du titre courant, de niveau 1, 2 ou 3
- un contenu textuel explicite (et invariable de page en page par exemple la date de fabrication du livre-esquisse) note : il peut contenir des variables ou du code à évaluer,
- une procédure évaluée à la levée

Définition des entêtes et pieds de page

Pour définir les entêtes et les pieds de page, le mieux est d'utiliser l'assistant, c'est le meilleur moyen de ne pas faire d'erreur pour cette donnée sensible et complexe.

Pour lancer l'assistant, jouer \$> pfb assistant et choisir "Assistant Header Footer".

Positionnement

Pour bien régler la position des headers et footers, il faut comprendre qu'ils s'inscrivent toujours par rapport à la marge définie, dans cette marge (l'idée est que la marge définit donc toujours la vraie surface contenu du texte, que rien ne vient la rogner — sauf les numéros de paragraphes lorsqu'ils sont utilisés).

Pour les entêtes, ils sont inscrits 5 PS-Points au-dessus de la marge haute. Il faut donc que cette marge haute fasse au moins 5 + <hauteur de ligne d'entête> (rappel : Prawn laisse toujours 10 PS-Points de fond perdu autour des pages).

Affiner le positionnement on joue sur la propriété header_vadjust et la propriété footer_vadjust de la disposition (qui se règle en ps-point). De cette manière, en jouant sur les marges hautes et basses et sur cette valeur, on peut avoir le positionnement exact désiré.

Note: la valeur, avec l'assistant, peut aller de -20 à 20. Si on doit utiliser une autre valeur (ce qui n'est pas conseillé...) éditer la recette à la main.

Tiers et contenus

Comme nous l'avons dit, on considère qu'un entête et un pied de page est divisé en deux fois trois "TIERS" occupant chacun un tiers de la largeur de la page, d'où leur nom. Pour définir un "headfooter", ces trois cases n'ont pas à être définis.

Ces tiers sont repérés par des clés qui portent en préfix l'indication de la page pg pour "page gauche" et pd pour "page droite" et en suffixe la position du tiers dans la page : _left pour le tiers à gauche, _center pour le tiers au center et _right pour le tiers à droite. On a donc :

```
:headers_footers:
 :headfooters:
    :HF0001:
      :id: :HF0001
      :name: Le headfooter en démo
      :font n style: "Times-Roman/normal"
      :size: 12
      :pg left:
        # ... définition... (il ne faut définir que les tiers utiles)
      :pg_center:
        # ... définition...
      :pg_right:
        :content: :titre1 # requise
        :align: :right
        :size: 40
        :font n style: "Geneva/italic"
        :casse: :min # ou :all_caps, :keep, :title
      :pd_left:
        # ... définition...
      :pd center:
        # ... définition...
      :pd right:
        # ... définition...
```

Dans la recette

```
# ...
#<headers_footers>
:headers_footers:
   :dispositions:
    # ... définition des dispositions (table)
:headfooters:
    # ... définition des headfooters (table)
```

Variables

On peut utiliser des variables à l'aide de #{nom_de_la_variable} dans un texte personnalisé.

Pages spéciales

Page de titre

La page de titre n'est pas à confondre avec la couverture (qui fait l'objet d'un fichier séparé pour un traitement différemment comme c'est souvent le cas). Il s'agit ici de la page, souvent après la page de faux titre et la page de garde qui présente toutes les informations générales sur le livre, titre, sous-titre, auteur, éditeur.

Pour sa mise en page, voir la <u>recette concernant les pages spéciales</u>.

Page d'informations

Nous appelons "page d'informations" la page de fin de livre où sont présentés toutes les informations sur la conception du livre, metteur en page, correcteurs, imprimeurs, isbn et autre date de dépôt légal.

Pour définir les informations, ouvrir une fenêtre de Terminal au dossier du livre ou de la collection et utiliser l'assistant en jouant la commande \$> pfb assistant et choisir "Assistant Page Infos".

Ces informations peuvent être réparties de 3 façons différentes :

- distribuées sur la page (réparties de façon égale sur la surface d'une page entière)
- en haut de page (toutes les informations sont rassemblées de façon compacte au-dessus d'une des dernières pages),
- en bas de page (toutes les informations sont rassemblées de façon compacte en bas d"une des dernières pages).

Table des matières

Recette	propriété	valeurs possibles
	:table_of_contents	Table de valeurs cf.

La table des matières se construit sur la base des titres.

Elle s'inscrit dans le livre à l'endroit où est placé dans le texte un :

```
(( toc ))
<!-- OU -->
(( tdm ))
```

"toc" signifie "Table of Contents" ou "Table des matières" en anglais.

ATTENTION: La construction de la table des matières n'ajoute pas automatiquement de nouvelles pages si la table déborde de la page qui lui est réservée (**FAUX***) (tout simplement parce qu'alors tous les numéros de pages seraient obsolètes...). Si la table des matières tient sur plusieurs pages, il faut donc ajouter autant de <u>marques de nouvelles pages</u> que voulus.

*En fait, elle le fait maintenant, mais si la pagination après, ça n'est pas trop grave ? Non, en fait, il faudrait que soit calculé dans le premier tour le nombre de page pour la table des matières et qu'elle soit inscrite ensuite. Noter que si la tdm est inscrite à la fin du livre, il n'y a plus de problème.

Voir la partie <u>Tous les types de pages</u> qui définit la recette du livre.

Voir ici pour exclure un titre de la table des matières.

Page d'index

Le plus simple pour construire un index dans un livre est d'utiliser la mise en forme par défaut, autant dans l'identification des mots à indexer que dans l'aspect de l'index final. Si l'on respecte ça, pour ajouter l'index, on a juste à insérer le texte suivant dans le texte du livre :

```
(( index ))
```

À l'endroit de cette marque sera inséré un index contenant tous les mots indexés dans le texte.

Par défaut, on repère les mots à indexer dans le texte par :

```
Ceci est un index:mot unique à indexer.

Ceux-là sont dans un index(groupe de mots) qu'il faut entièrement indexer.

Ce index(mot|verbe) doit être indexé avec le mot "verbe" tandis que :

Ces index(mots-là|idiome) doivent être indexé avec le mot "idiome".

# La barre "|" sert souvent pour séparer les données dans PFB.
```

Si l'on veut utiliser une autre méthode pour indexer les mots, on peut définir la méthode paragraph parser (paragraph du fichier parser du livre ou de la collection.

cf. <u>Parsing personnalisé du texte</u> 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)

# Note : @table_index a déjà été initiée avant
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</pre>
```

À l'issue du traitement, la table <code>@table_index</code> (de l'instance <code>PdfBook</code>) contiendra en clé tous les mots trouvés et en valeur une liste de toutes les itérations. Cette liste contiendra la liste des pages ou la liste des paragraphes en fonction du type de pagination adopté pour le livre ou la collection.

On peut donc faire ensuite:

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

BIBLIOGRAPHIES

Voir la partie <u>Tous les types de pages</u> qui définit la recette du livre pour avoir un aperçu rapide des la définition d'une bibliographie.

On peut obtenir un assistant à la définition des bibliographies du livre ou de la collection en jouant la commande :

```
$> pfb aide biblio
```

Une bibliographie nécessite :

- de <u>définir la balise</u> qui va repérer les éléments dans le texte (par exemple film ou livre)
- de <u>définir un titre</u> qui sera utilisé dans le livre (:title ou clé définie par :main_key),
- de <u>définir le **chemin d'accès**</u> à ses données (:path),
- de définir la page sur laquelle sera écrite la bibliographie,
- de <u>définir les données</u> utilisées par la bibliographie et qu'elles soient valides,
- de <u>définir la mise en forme</u> utilisée pour le livre pour présenter les informations sur les éléments.

La balise de la bibliographie

Recette	propriété	valeurs possibles
	:bibliographies:	null/table

La balise est le mot qui sera utilisé pour repérer dans le texte les éléments à ajouter à la bibliographie. Par exemple, pour une liste de films, on pourra utiliser film:

```
Je vous parle d'un film qui s'appelle film(idFilmTitatic|Le Titanic) et se déroule dans un bateau.
```

Elle est définit dans la propriété : tag dans le livre de recette du livre ou de la collection :

```
# in recipe.yaml
# ...
:bibliographies:
:book_identifiant: 'livre'
:biblios:
   film:
     # ...
```

Dans le texte, elle doit définir en premier argument l'identifiant de l'élément concerné dans les données.

Cette balise permettra aussi de définir la bibliographie à inscrire dans le livre, sur la page voulue, avec la marque :

```
(( bibliographie(film) ))
ou
(( bibliography(film) ))
ou
(( biblio(film) ))
```

Pour plus de détail, cf. <u>la page de la bibliographie</u>

Le titre de la bibliographie

Ce titre est celui qui apparaitra sur la page de bibliographie du livre. Il doit être défini entièrement, par exemple "Liste des films cités" ou "Liste des livres utiles".

Il est défini par la propriété :title dans la recette du livre ou de la collection.

```
# in recipe.yaml
# ...
:bibliographies:
:biblios:
   film:
   :title: Liste des films cités
```

Par défaut, ce titre sera d'un niveau 1, c'est-à-dire d'un niveau grand titre. Mais on peut définir son niveau propre à l'aide de :title_level:

```
# in recipe.yaml
# ...
:bibliographies:
:biblios:
film:
    :title: Liste des films cités
:title_level: 3
```

Chemin d'accès aux données de la bibliographie

L'autre donnée absolument requise pour qu'une bibiographie soit opérationnelle concerne son <code>:path</code>, c'est-à-dire le chemin d'accès à ses données, donc le dossier contenant les fiches de ses items.

```
# in recipe.yaml
# ...
:bibliographies:
:biblio:
    mabib:
    :title: Le Titre de MaBib
    :path: ./path/to/cards/folder
```

Comme on peut le voir, ce chemin peut être défini de façon relative (par rapport au dossier du livre, ou de façon absolue (ce qui n'est pas recommandé, si le dossier change de place plus tard ou si le dossier du livre est transmis..

La page de la bibliographie

On utilisera simplement la marque suivante pour inscrire une bibliographie sur la page :

```
(( biblio(<tag>) ))
ou (( bibliographie(<tag>) ))
ou (( bibliography(<tag>) ))
```

... où <tag> est la balise définie dans la recette du livre (propriété :tag).

Une bibliographie ne s'inscrit pas nécessairement sur une nouvelle page. Si ça doit être le cas, il faut placer le code ((new_page)) avant.

```
# in recipe.yaml
# ...
:bibliographies:
:biblios:
  film:
    :title: Liste des films
    :title_level: 2
```

Noter que si le niveau de titre est 1 (ou non défini), et que les propriétés des titres de la recette définissent qu'il faut passer à une nouvelle page pour un grand titre, la bibliographie commencera alors automatiquement sur une nouvelle page.

Les données de la bibliographie

Les données bibliographiques sont contenus dans un dossier, par fiche (une fiche par item bibliographique) au format yaml ou json.

La source des données (le dossier) est indiquée dans le fichier recette du livre ou de la collection par la propriété :path :

```
# in recipe.yaml
# ...
:bibliographies:
:biblios:
    film: # le tag singulier
        :title: Liste des films
        :title_level: 2
        :path: data/films
        :main_key: :titre_fr # pour définir une autre clé par défaut
        :font: Fonte # la fonte à utiliser
        :size: 10 # la taille de fonte (10 par défaut)
        :style: null # éventuellement le style de la fonte
```

Ci-dessus, la source est indiquée de façon relative, par rapport au dossier du livre ou de la collection, mais elle peut être aussi indiquée de façon absolue si elle se trouve à un autre endroit (ce qui serait déconseillé en cas de déplacement des dossiers).

Pour le moment, *Prawn-for-book* ne gère que les données au format YAML et JSON. Ces données doivent produire une table où l'on trouvera en clé l'identifiant de l'élément et en valeur ses propriétés, qui seront utilisées pour la bibliographie. Par exemple, pour un fichier films.yaml qui contiendrait les données des films:

```
# in data/films/titanic.yaml
---
title: The Titanic
title_fr: Le Titanic
annee: 1999
realisateur: James CAMERON

# in data/films/ditd.yaml
---
title: Dancer in The Dark
annee: 2000
realisateur: Lars VON TRIER

# etc.
```

NOTE IMPORTANTE: toute donnée bibliographique doit avoir une propriété :title ou la propriété définie par :main_key dans la définition de la bibliographie, qui sera écrite dans le texte à la place de la balise.

Note: mais ce comportement peut être surclassé en implémentant la méthode

FormaterBibliographieModule::<id biblio>_in_text(data) qui reçoit la table des données de l'élément tel qu'il est enregistré dans sa fiche.

Voir ensuite dans <u>la partie mise en forme</u> la façon d'utiliser ces données.

Mise en forme des données bibliographiques

La mise en forme des bibliographies (ou de *la* bibliographie) doit être définie dans le <u>fichier formater.rb</u>.

Attention : ici, nous parlons bien de la mise en forme des éléments dans la bibliographie, par exemple en fin d'ouvrage. Il ne s'agit pas de la mise en forme dans le texte, qui sera abordée plus bas.

Il faut y définir une méthode préfixée biblio_suivi par la balise (:tag) de la bibliographie concernée. Ce sera par exemple la méthode biblio film pour la liste des films.

```
# in formater.rb
module BibliographyFormaterModule
 # Méthode mettant en forme les données à faire apparaitre et renvoyant
  # le string correspondant.
  # @param \Bibliography::BibItem element L'élément bibliographique
 # @param \PrawnView
 def biblio_tt(element, pdf)
   c = []
   element.instance eval do
      c << title
      c << " (#{title_fr})" if title_fr</pre>
      c << annee
    end
    return c.join(', ')
  end
  # Autre tournure possible
 def biblio tt(element, pdf)
    TEMPLATE TT % element.data
  end
  TEMPLATE_TT = '%{title} de %{writers}, %{year}'.freeze
end #/module BibliographyFormaterModule
```

Noter qu'avec cette formule, les données sont toujours présentées sur une ligne. Mais on peut, grâce aux <u>tables</u>, obtenir un affichage très précis. Voir l'<u>exemple plus haut</u>.

Noter également qu'on n'indique pas, ici, les pages/paragraphes où sont cités les éléments, cette information est ajoutée automatiquement par l'application, après le titre et deux points. L'indication par page ou par paragraphe dépend du type de <u>pagination</u> adoptée dans le livre. En conclusion, le listing final ressemblera à :

```
<partie définie par biblio_tag> : <liste des pages/paragraphes séparés par des
virgules>.
<partie définie par biblio_tag> : <liste des pages/paragraphes séparés par des
virgules>.
<partie définie par biblio_tag> : <liste des pages/paragraphes séparés par des
virgules>.
```

Mise en forme de l'item de bibliographie dans le texte

La section précédente parlait de la mise en forme de la bibliographie elle-même, souvent placée à la fin du livre. On peut également définir comme l'item apparaitra dans le texte lui-même de façon très fine et très complexe.

On le fait grâce à la méthode <tag biblio> in text dans le fichier formater.rb :

```
module BibliographyFormaterModule
 # @param [Bibliography::BibItem] instance Item bibliographique. On peut
                     obtenir toutes ses données par instance.property
 # @param [Hash] context Le contexte d'écriture (pour connaitre la page,
                            le paragraphe, la taille de police, etc.
 # @option context [AnyParagraph] paragraph L'instance du paragraphe. Typiquement
                      pour être en mesure de modifier paragraph.final_specs
  #
                      qui permet d'ajouter des traitements à pdf.text
                      Mais ATTENTION: ces traitements touchent tout le para-
  #
                      graphe, pas seulement le mot. Si, par exemple, on définit
  #
                      paragraphe.final_specs.merge!({kerning:true, \
  #
                      character_spacing: -1}) alors ce sont tous les mots du
                     paragraphe qui seront modifiés.
 # @param [String] actual Optionnellement, le mot à écrire (par exemple le
  #
                           mot au pluriel.
 # @return [String] Le texte à afficher
 def film in text(instance, context, actual)
   # traitement à partir des +data+ de l'item
   return texte_a_afficher
 end
```

Rappel: en utilisant le stylage inline, on peut modifier considérablement l'aspect des mots de bibliographie.

À titre d'exemple de complexité, on présente ci-dessous le traitement d'une bibliographie de livres, avec trois affichages différents en fonction du contexte. Lorsque le livre est cité pour la première fois dans le texte, il contient toutes ses informations (c'est-à-dire, ici, son titre, son auteur et son année de publication). Pour une citation suivante, mais assez proche (moins de 10 pages), on ne met que le titre du livre, en

italiques. Si, enfin, le livre est à nouveau cité, mais à plus de 10 pages, on ajoute l'année de publication à nouveau et seulement le nom.

```
module BibliographyFormaterModule
 ##
 # Formatage d'une citation de livre dans le livre en exemple
 # Il y a trois formatages différents :
 # - la première fois où le livre est cité
    - une autre fois proche de la dernière citation (< 10 pages)
     - une autre fois loin de la dernière citation (> 10 pages)
 def self.livre_in_text(livre, context, actual)
   pa = context[:paragraph]
   # Pour consigner les informations des livres déjà cités
   @@livres_cites | = {}
   # true si c'est la toute première citation du livre
   premiere_fois = not(@@livres_cites.key?(livre.id))
   # true si le livre a été cité la dernière fois à plus
   # de 10 pages de là
   citation_lointaine = not(premiere_fois) && (pa.first_page >=
@@livres cites[livre.id][:last page] + 10)
   str = "<i>#{livre.title}</i>"
   # @note
   # 'title' peut être la propriété définie par :main_key dans la
       définition de la bibliographie.
   if premiere_fois
     @@livres_cites.merge!(livre.id => {last_page: pa.first_page})
     str = "#{str} (#{livre.auteur}, #{livre.annee})"
   else
     if citation_lointaine
        str = "#{str} (#{livre.auteur_last_name}, #{livre.annee})"
     end
     # On consigne la dernière utilisation
     @@livres_cites[livre.id][:last_page] = pa.first_page
   return str
  end
```

```
# Extension de BibItem pour obtenir le patronyme de l'auteur
  # seulement (écriture façon univsersité).
  class Prawn4book::Bibliography::BibItem
    def auteur last name
      @auteur_last_name || = begin
        dauteur = auteur.split(' ')
        if dauteur.count == 2
          dauteur[1]
        else
          dauteur.select do | mot |
            mot.upcase == mot
          end.join(' ')
        end
      end
    end
  end
end #/module BibliographyFormaterModule
```

Noter comment on peut étendre la classe Prawn4book::Bibliography::BitItem pour définir de nouvelle méthodes/propriétés pratiques.

RÉFÉRENCES (et références croisées)

On peut faire très simplement des références dans le livre (références à d'autres pages ou d'autres paragraphes, du livre ou d'autres livres) à l'aide des balises :

```
(( <-(id_reference_unique) )) # référence (cible)
(( ->(id_reference_unique) )) # appel de référence
```

La référence sera tout simplement supprimée du texte (attention de ne pas laisser d'espaces — même si, normalement, ils sont supprimés). Pour l'appel de référence il sera toujours remplacé par "la page xxx" ou "le paragraphe xxx" en fonction de <u>la pagination souhaitée</u> et du préfix de référence choisi (TODO).

Références croisées

Pour une *référence croisée*, c'est-à-dire la référence à un autre livre, il faut ajouter un identifiant devant la référence et préciser le sens de cet identifiant. Elle peut ressembler à :

```
Rendez-vous sur la (( ->(IDLIVRE:id_ref_uniq) )).
```

Par exemple:

```
Rendez-vous sur la (( ->(mon_livre:sa_reference) )).
```

... qui sera transformé en :

```
Rendez-vous sur la page 12 de <i>Mon beau livre référrencé</i>.
```

Ci-dessus, IDLIVRE est l'identifiant du livre (tel que défini dans la bibliographie (des livres) et id_ref_uniq est une référence définie pour le livre.

Pour traiter une référence croisée, on a besoin de plusieurs choses :

• que la bibliographie des livres soit définie dans le fichier recette (du livre ou de la collection :

```
# ...
#<bibliographie>
bibliographies:
  biblios:
    livre:
     title: Liste des livres
     path: livres.yaml
#</bibliographie>
```

• que le livre en question soit défini dans les données bibliographiques (le fichier livres.yaml cidessus, qui se trouve donc à la racine du dossier du livre — ou de la collection), avec, en plus des autres livres, une définition du fichier contenant les références :

```
# dans ./livres.yaml
---
# ... des livres
mon_livre:
   title: Mon beau livre référencé # le titre qui sera repris
   refs_path: resources/mon_livre.references.yaml
   auteur: # etc.
```

Notez que le livre peut ou peut ne pas être un livre prawn-for-book. Dans le cas d'un livre prawn-for-book, le fichier s'appellera toujours references yaml et sera toujours placé à la racine du dossier du livre. On pourra avoir alors :

```
# dans ../livres.yaml (dossier de la collection)
---
# ... des livres
livre_collection_1:
    titre: "Premier tome de la collection"
    refs_path: "tome1/references.yaml"
    # etc.
livre_collection_2:
    titre: "Deuxième tome de la collection"
    refs_path: "tome2/references.yaml"
    # etc.
```

 que les références soient bien définies avec, en clé, leur identifiant et en valeur la page et le paragraphe cibles. Par exemple :

```
# dans ./tome1/references.yaml
---
sa_reference:
  page: 12
  paragraphe: 59
autre_reference:
  page: 15
  paragraphe: 78
# etc.
```

Notez quand même, ci-dessus, que s'il s'agit d'un livre prawn-for-book, ce fichier de références est automatiquement produit, donc il n'y pas besoin de le faire.

Noter qu'il peut être difficile de connaitre le numéro de paragraphe dans un livre imprimé. Dans ce cas, laisser la donnée vide et, si les références se font par paragraphe, c'est exceptionnellement la donnée page qui sera utilisée).

Exclure des paragraphes

Cf.ci-dessous les commentaires.

Commentaires

Pour ajouter des commentaires dans un fichier texte destiné à l'impression, on le place entre commentaire markdown normaux.

```
<!-- Commentaire sur une ligne -->

<!--
Commentaires
Sur plusieurs
Lignes
-->
```

Il est donc tout à fait possible d'exclure du texte en le mettant entre ces signes :

```
# Titre principal
<!--
## Titre zappé
Paragraphe zappé, non imprimé
-->
## Un titre pris en compte.
```

Noter que par rapport à du markdown pur, il est inutile de laisser des lignes vierges entre les types de paragraphes.

PARSE(U)RS & FORMATE(U)RS

Utilisation expert, connaissances en programmation ruby requise.

Les *parseurs* et les *formateurs* sont un moyen puissant pour mettre en forme de façon homogène tout le texte du livre.

À titre d'exemple, voici un code qui utilise leur puissance :

```
Je connais gens(John) mais aussi gens(Mathilde) et même gens(René).
```

J'utilise ici la tournure identifiant (contenu) mais on peut utiliser n'importe quelle forme qui peut être reconnue par une expression régulière.

Il me suffit ensuite de définir, dans le module parser.rb:

```
# in parser.rb (collection ou livre)

module ParserParagraphModule
  class Personne
    @@personnes = []
  end

# Cette méthode sera automatiquement appelée avec le texte du
# paragraphe.
  def self.parser_formater(str, pdf)

#
# Ici, on analyse le texte du paragraphe est on le transforme
#
str = str.gsub(/gens\((.+?)\)/) do
  lapersonne = $1.freeze

#
# Je peux même la mettre de côté pour une utilisation future
#
Personne.personnes << lapersonne
#
# Je mets en forme le texte. "gens(quelqu'un)" sera remplacé par
# "quelqu'un" dans une autre police de caractère.</pre>
```

```
#
  "<font name='Geneva'>#{lapersonne}</font>"
  end
  return str
end
```

Prawn-for-book utilise 3 moyens de travailler avec les paragraphes au niveau du code :

- un module de formatage personnalisé (formater.rb),
- un module de méthodes d'helpers qui permettent un traitement ruby personnalisé (helpers.rb),
- un module de méthode de parsing qui traite de façon propre le paragraphe (parser.rb).

Ces trois fichiers (parser.rb, helpers.rb et formater.rb) sont propres à chaque livre ou chaque collection et seront toujours automatiquement chargés s'ils existent.

Méthode d'erreurs (gestion des erreurs)

Dans tous les modules ruby, on peut ajouter des erreurs mineures — qui n'interrompront pas la fabrication du livre, mais génèreront une erreur à la fin — à l'aide de la méthode de haut niveau add_erreur(err, options).

Par exemple:

```
def balise_parser(str)
   str.length < 10 || add_erreur("La balise #{str.inspect} devrait faire moins de 10
   signes.")
end</pre>
```

Message de notification

Comme pour la <u>gestion des erreurs</u>, on peut produire des messages « notice » à la fin du processus de fabrication du livre à l'aide de la méthode <u>add_notice(msg, **options)</u>.

```
def affiche_image
  if image.exist?
    add_notice("Pour actualiser l'image, la détruire.")
  else
    fabrique_image
  end
# ...
end
```

Méthode d'helpers

Les méthodes d'helpers s'utilisent dans le texte comme un code ruby :

```
Ceci est un texte de paragraphe avec un (( #code_ruby_simple )) qui sera évalué.

Ceci est un paragraphe avec qui devra apprendre à dire (( #code_ruby("bonjour tout le monde") )).
```

Attention : ne pas oublier les espaces à l'intérieur des parenthèses, comme c'est le cas avec le signe de Prawn, les doubles parenthèses.

Au besoin, cette méthode d'helpers peut générer des pages entières.

Cette méthode ou variable code_ruby_simple doit être définie en *Ruby* dans le fichier helpers.rb du [livre][] ou de la [collection][] de la manière suivante :

```
# in ./dossier/livre/helpers.rb
module PrawnHelpersMethods
def code_ruby_simple
    # On utilise ici 'pdfbook' et 'pdf' pour obtenir le livre ou
    # son builder.
    # On retourne la position actuelle du curseur dans le fichier
    # pdf en l'arrondissant :
    return round(pdf.cursor)
end

def code_ruby(str)
    return "« #{str} »"
end
end
```

Ces méthodes d'helpers doivent obligatoirement retourner le code (le texte) qui sera écrit à leur place dans le paragraphe.

Les seules conventions a respecter ici sont :

- le fichier doit impérativement s'appeler helpers.rb (au pluriel, car il y a plusieurs helpers mais l'application cherchera aussi le singulier),
- le fichier doit impérativement se trouver à la racine du dossier du livre ou du dossier de la collection (les deux seront chargés s'ils existent attention aux collisions de noms),
- le titre du module doit être **PrawnHelpersMethods** (noter les deux au pluriel et là c'est impératif).

Les méthodes ont accès à **pdfbook** et **pdf** qui renvoient respectivement aux instances

Prawn4book::PdfBook et Prawn4book::PrawnView. La première gère le livre en tant que livre (pour obtenir son titre, ses auteurs, etc.) et la seconde est une instance de Prawn::View (substitut de Prawn::Document) qui génère le document PDF pour l'impression.

On peut par exemple obtenir le numéro de la page avec pdf.page_number et la consigner :

```
Ceci est un paragraphe avec au bout un code qui sera caché (remplacé par un string vide) pour savoir le numéro de cette page et le numéro de ce paragraphe.(( #consigne_page('page_a_memoriser') ))(( #consigne_paragraphe('par2memo') ))
```

... avec les deux méthodes d'helpers définies ainsi :

```
# in helpers.rb
module PrawnHelpersMethods

def consigne_page(id)
    @pages_memorisees ||= {}
    @pages_memorisees.merge!(id => pdf.page_number)
    return ''
end

def consigne_paragraphe(id)
    @paragraphes_memorises ||= {}
    @paragraphes_memorises.merge!(id => pdf.paragraph_number)
    return ''
end
end
```

Grâce à pdfbook, on a accès à l'intégralité des valeurs de la recette. Ce qui signifie qu'on peut consigner n'importe quelle valeur dans la recette, qu'on pourra récupérer dans ces helpers. Par exemple, si on définit dans la recette :

```
# in recipe.yaml
---
# ...
:ma_couleur_preferee: '#2569F8'
:une_autre_couleur: '#45DF56'
```

... alors on pourra utiliser dans le helper :

```
module PrawnHelpersMethods
# @return le texte +str+ en le mettant à la couleur +which_color+ qui est
# une couleur hexa définie dans la recette du livre
def colorise(str, which_color)
    code_couleur = pdfbook.recipe.get(which_color)
    return "<font color=\"#{code_couleur}\">#{str}</font>
    end
end
```

... et l'utiliser dans le texte avec :

```
Ce paragraphe contient un (( #colorise("texte", :ma_couleur_preferee) )) qui sera dans ma couleur préférée et un (( colorise("autre texte", :une_autre_couelur) )) qui sera dans une autre couleur.
```

Ce texte, une fois construit, produira:

TODO: montrer l'image produite.

Formatage personnalisé (formater.rb)

Formatage des paragraphes

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

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

```
# in ./formater.rb
module FormaterParagraphModule # Ce nom est absolument à respecter
# @note
# __formate_custag est une méthode d'instance du paragraphe, qui
# a donc accès à toutes ses propriétés, dont @text qui contient le
# texte.
def __formate_custag
# ...
@text = transformation_de_la_propriete(text)
end
end

module BibliographyFormaterModule # ce nom est absolument à respecter
end #/module
```

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 contiennent ce fichier, **les deux seront chargés** ce qui permet d'avoir des formateurs propres à la collection complète et d'autres propres aux livres en particulier.

Un formatage classique consiste à appliquer une police, taille et style particulière au texte. Par exemple, si on trouve dans le texte :

```
Ceci est un paragraphe normal.

style1::Ce paragraphe est stylé par le premier style.

Un autre paragraphe normal.
```

... alors on peut avoir dans le fichier formater.rb de la collection :

```
# Dans collection/formater.rb
module FormaterParagraphModule
  LINE_FORMATED = '<font name="Arial" size="40" style="bold italic">%s</font>'
  def __formate_style1
    @text = LINE_FORMATED % text
  end
end
```

... qui va appliquer la police Arial, les styles gras et italique et la taille 40 au texte.

Insertion d'un titre par un helper

Pour insérer un titre de façon programmatique par le biais d'un helper, utiliser la formule suivante :

```
def _mon_helper # voir ci-dessus pour le nom

# Définition du titre
  titre = Prawn4book::PdfBook.NTitre.new(pdfbook, {level: <niveau du titre>, text: "
  <titre>"})
  # Écrire dans le livre
  pdf.start_new_page
  titre.print(pdf)

end
```

Note: on peut souvent obtenir pdfbook par pdf.pdfbook. En dernier recours, si vraiment il n'est pas accessible, on peut utiliser Prawn4book::PdfBook.current.

Tailles de police proportionnelles

Une des limites de **Prawn** est de ne pas pouvoir utiliser les unités proportionnelles comme em ou rem. Pour adapter une police au contexte — lorsqu'elle ne garde pas sa taille normale —, il faut donc utiliser un détour.

Pour ce faire, on va utiliser un rapport à utiliser dans la police.

Imaginons que nous utilisons une balise perso(Selma) pour marquer de façon différente les personnages dans livre analysant un film. Cette balise utilise la police Bangla qui doit être réduite pour s'adapter correctement à la police Garamond du livre. Nous savons que sa taille doit être 8.75 lorsque Garamond fait 12. Le rapport est donc 8.75 / 12.

On pourra alors utiliser ce code:

```
# Dans parser.rb
module ParserParagraphModule
 def __paragraph_parser(paragraph, pdf)
   paragraph.text = parser_formater(paragraph.text, pdf
 end
  # @note
  # Traitement séparé pour être appelé par les textes seuls, comme dans les
  # table
  def parser formater(str, pdf)
    # On calcule une seule fois le rapport
    @rapport_size_perso ||= 8.75 / 12
    # On peut calculer la taille courante dans la balise grâce
    # à la taille courante dans le pdf
    size_perso = (@rapport_size_perso * pdf.current_font_size).round(2)
    # On remplace dans le texte
    str = str.gsub(REG PERSO){ SPAN PERSO % [size perso, $1] }
    return str
   end
  REG_PERSO = /perso\((.+?)\)/.freeze
  SPAN_PERSO = '<font name="Bangla" size="%s">%s</font>'.freeze
end #/module ParserParagraphModule
```

Formatage des éléments de bibliographie

Le formatage est défini dans des méthodes biblio_<tag> dans un module BibliographyFormaterModule du fichier formater.rb:

```
# in formater.rb

module BibliographyFormaterModule
  def biblio_film(film)
    # ...
  end
end
```

Formatage des tables

Pour le formatage propre des tables, cf. <u>Définir un style de table</u>.

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

Présentation générale

La recette du livre (ou de la collection) permet de définir tous les aspects que devra prendre le livre, c'est-àdire le fichier PDF prêt-à-imprimé, dans le moindre détail. On définit dans ce fichier les polices utilisées (à empaqueter), les marges et la taille du papier, les titres, les lignes de base, le titre, les auteurs, le décalage du chiffrage du paragraphe, le contenu des entêtes par section, etc.

Création de la recette du livre

Le plus simple pour créer la recette d'un livre est d'<u>utiliser l'assistant de création</u>.

Cet assistant permet de créer le fichier recipe.yaml contenant la recette du livre (ou recipe_collection.yaml pour la recette de la collection.

Une recette de collection permet de répéter les mêmes recettes pour tous les livres de la collection en question.

Contenu de la recette du livre

Dans la partie suivante est présentée l'intégralité des propriétés définissables dans le fichier recette du livre ou de la collection.

Éléments de la recette

book_data (informations générales du livre)

Si ces informations sont rentrées à la main, ne pas oublier les balises-commentaires (#<book_data>) qui permettront d'éditer les données.

```
# in recipe.yaml

#<book_data>
book_data:
    title: "Titre du livre"
    subtitle: "Sous-titre\nSur plusieurs\nLignes"
    id: "identifiant_livre" # utile
    auteurs: ["Prénom NOM", "Prénom DE NOM"]
    isbn: "128-9-25648-635-8"

#</book_data>
```

collection_data (données pour la collection)

```
# Dans collection_recipe.yaml

#<collection_data>
collection_data:
   name: "Nom humain de la collection"
   short_name: "Nom raccourci" # pour les messages seulement
#</collection_data>
```

book_format (format du livre)

```
# in recipe.yaml/collection recipe.yaml
#<book format>
book_format:
 book:
   width: "125mm"
   height: "20.19cm"
   orientation: "portrait"
 page:
   margins:
     top: "20mm" # marge haute
     bot: 50 # marge basse
     ext: "2cm"
                  # marge extérieure
     int: "0.1in" # marge intérieure
   numerotation: "pages" # ou "parags"
   pagination format: "first-last" # pour numérotation paragraphes
   no num if empty: true # pas de numéro sur pages vides
   num only if num: true # cf. [1]
   num page if no num parag: true # cf. [2]
   no_headers_footers: false # self-explanatory
   skip_page_creation: true # cf. [3]
  text:
   default_font_and_style: "Helvetica/normal"
   default size: 11.2
   line height: 14 # hauteur de ligne cf. [4]
   indent: 0 # indentation
   leading: 0 # espace entre paragraphes
   parag_num_vadjust: 1 # ajustement vertical numéro paragraphe
   parag_num_dist_from_text: 0 # ajustement horizontal
   parag_num_size: 9 # taille de la police pour les numéros de paragraphe
   parag_num_strength: 72 # "force" (opacité) du numéro de paragraphe
#</book format>
```

[1]

On ne met un nombre que si réellement il y a un nombre. Par exemple, si c'est une numérotation par paragraphe et que la page ne contient aucun paragraphe, cette page n'aura pas de paragraphe (sauf si l'option :num_page_if_no_num_parag est activée, bien sûr.

[2]

Si :numerotation est réglé sur 'parags' (numérotation par les paragraphes) et qu'il n'y a pas de paragraphes dans la page, avec le paramètres :num_page_if_no_num_parag à true, le numéro de paragraphe sera remplacé par le numéro de la page.

À la création (génération) d'un livre avec Prawn, une page est automatiquement créée. On peut empêcher ce comportement en mettant ce paramètre à true.

[4]

line_height est un paramètre particulièrement important puisqu'il détermine la <u>grille de référence</u> du livre qui permet d'aligner toutes les lignes, comme dans tout livre imprimé digne de ce nom.

[5]

Ajustement de la position du numéro de paragraphe (si c'est une numérotation par paragraphe). D'abord verticalement, puis horizontalement en distance par rapport au texte.

titles (données d'affichage des titres)

```
# in recipe.yaml/collection_recipe.yaml
#<titles>
titles:
 # Données pour le titre du niveau 1
 level1:
   next page: true # sur nouvelle page ?
   belle page: false # toujours sur belle page ?
   :font_n_style: "LaFonte/lestyle"
                  30
   :size:
   :lines_before: 0  # cf. [1] [3]
:lines_after: 4  # cf. [1]
   :leading: -2 # interlignage cf. [2]
  :level2:
   # idem
  :level3:
   # idem
  # etc.
#</titles>
```

[1]

Les **lines_before** et **lines_after** se comptent toujours en nombre de lignes de référence, car les titres sont toujours alignés par défaut avec ces lignes (pour un meilleur aspect). On peut cependant mettre une valeur flottante (par exemple 2.5) pour changer ce comportement et placer le titre entre deux <u>lignes</u> de référence.

[2]

La valeur du **leading** permet de resserrer les lignes du titre afin qu'il ait un aspect plus "compact", ce qui est meilleur pour un titre. Ne pas trop resserrer cependant.

[3]

le :line_before d'un titre suivant s'annule si le titre précédent en possède déjà un. Si par exemple le titre de niveau 2 possède un :lines_after de 4 et que le titre de niveau 3 possède un :lines_before de 3, alors les deux valeurs ne s'additionnent pas, la première (le :lines_after du titre de niveau 2) annule la seconde (le :lines_before du titre de niveau 3).

Bien noter que c'est vrai dans tous les cas. Par exemple, si un titre de niveau 1 a son :lines_after réglé à 0, un titre de niveau supérieur aura beau avoir son :lines_before réglé à 4 ou 6, le titre de niveau supérieur sera "collé" au titre de niveau 1.

Par défaut, les titres (leur première ligne, s'ils tiennent sur plusieurs lignes) se placent toujours sur des lignes de référence.

publishing (données de la maison d'édition)

```
# in recipe.yaml/collection_recipe.yaml
#<publishing>
publishing:
 name:
            "Nom édition" # p.e. "Icare Éditions"
 adresse: "Numéro Rue\nCode postal Ville\nPays
 url:
            "https://site-des-editions.com"
 logo_path: "path/to/logo.svg" # cf. [1]
 siret:
             "NUMEROSIRET"
 mail:
             "info@editions.com"
                                 # mail principal
 contact:
            "contact@editions.com" # mail de contact
  [2]
#</publishing>
```

[1]

Ce doit être le chemin d'accès absolu (déconseillé) ou un chemin relatif dans le dossier du livre OU le dossier de la collection.

[2]

On peut tout à fait ajouter toutes les informations supplémentaires que l'on voudra, le nom de l'éditeur par exemple. On pourra y faire référence ensuite, dans le livre, à l'aide de recipe.publishing[<key>]. Par exemple :

```
publishing:
    # ...
# publisher: Gaston GALLIMARD
```

Dans le texte, on pourra y faire référence par :

Le plus connu des éditeurs est sans doute #{recipe.publishing[:publisher]} créateur de la célèbre maison d'éditions de même nom.

fonts (données des polices)

```
# in recipe.yaml/collection_recipe.yaml
#<fonts>
fonts:
  fontName: # le nom de la police cf/ [1]
                "/path/to/font.ttf" # Style cf. [2][3]
   monstyle:
   autrestyle: "/path/to/font-autrestyle"
   bold:
                 "/path/to/bold.ttf" # [2]
   italic:
                 "/path/to/italic.ttf" # [2]
   bold italic: "/path/to/bold italic.ttf" # [2]
  autrePolice:
   monstyle: "..."
   # etc.
#</fonts>
```

[1]

C'est le nom que l'on veut, qui servira à renseigner les paramètres *font_n_style* des différents éléments. Par exemple, si le <code>font_n_style</code> d'un titre de niveau 2 est "MonArial/styletitre" alors la fonte "MonArial" doit être définie avec le path du fichier ttf à utiliser pour le style <code>styletitre</code>:

```
fonts:
    MonArial:
    styletitre: "/Users/fontes/Arial Bold.ttf"
```

[2]

Comme on le voit ci-dessus, on peut utiliser n'importe quel nom de style, pourvu qu'il soit associé à un fichier ttf existant. Cependant, certains noms de styles sont importants pour gérer correctement les balises de formatages HTML de type <i> ou . Pour <i>, il faut définir le style italic: et pour il faut définir le style italic: et pour il faut définir le style ibold.

[3]

Pour trouver le chemin d'accès à une police quelconque, voir la procédure décrite en annexe.

Dossiers des fontes

On peut définir les dossiers des fontes par variables pour y faire référence plus facilement. Mais notez que cette utilisation fonctionne seulement pour une définition des polices « à la main ». [Expert] Pour les définir avec l'assistant, il faut mettre les mains dans le cambouis (dossier de l'application) et définir les dossiers dans la constante **DATA_FONTS_FOLDERS** du fichier ./lib/commandes/assistant/assistants/fontes.rb.

```
# ...
# Une variable pour simplifier
dossier_fonts: &dosfonts "/Users/phil/Library/Fonts"
```

```
fonts system: &sysfonts "/System/Library/Fonts"
prawn_fonts: &pfbfonts "/Users/phil/Programmes/Prawn4book/resources/fonts"
# Définition des fontes (note : ce sont celles par défaut quand on
# utilise les templates)
#<fontes>
fonts:
  Garamond:
    :normal: "*dosfonts/ITC - ITC Garamond Std Light Condensed.ttf"
    :italic: "*dosfonts/ITC - ITC Garamond Std Light Condensed Italic.ttf"
 Bangla:
    :normal: "*sysfonts/Supplemental/Bangla MN.ttc"
           "*sysfonts/Supplemental/Bangla MN.ttc"
 Avenir:
    :normal: "*sysfonts/Avenir Next Condensed.ttc"
 Arial:
    :normal: "*dosfonts/Arial Narrow.ttf"
 Nunito:
    :normal: "*pfbfonts/Nunito Sans/NunitoSans-Regular.ttf"
            "*pfbfonts/Nunito_Sans/NunitoSans-Bold.ttf"
#</fontes>
```

L'ordre de définition des fontes ci-dessous peut être défini avec soin, car si certains éléments du livre ne définissent pas leur fonte, cette fonte sera choisie parmi les fontes ci-dessus. Pour des textes importants (comme les index, la table des matières, etc.) c'est la première fonte qui sera choisie tandis que pour des textes mineurs (numéros de paragraphes, entête et pied de page, etc.), c'est la seconde qui sera choisie.

bibliographies (données bibliographiques)

Voir ici pour le détail du fonctionnement et de la définition des bibliographies.

Rappel: le terme « bibliographies », ici, est très général, il peut concerner bien d'autres choses que des livres. Il peut concerner tout élément du texte que l'on veut rassembler, en fin de livre, sur une ou plusieurs pages, pour pouvoir y faire référence plus facilement.

On peut, par exemple, faire une bibliographie des personnages de l'histoire. À la fin du livre, on trouvera alors la liste de tous les personnages, avec le numéro des pages où ils apparaissent.

```
# in recipe.yaml/collection_recipe.yaml

#<bibliographies>
bibliographies:
  book_identifiant: "livre" # cf. [1]
  font_n_style: "Times-Roman/normal" # Fonte par défaut
  # Définition de toutes bibliographies utilisées
  biblios:
  letag: # par ex. "livre" ou "film" cf. [2]
       title: "Titre à donner à l'affichage" # cf. [3]
```

```
path: "path/to/dossier/fiches/or/fichier
  title_level: 1 # niveau de titre cf. [3]
  new_page: true # pour la mettre sur une nouvelle page cf. [3]
  font_n_style: null # ou la "Police/style" des items
  size: null # par défaut ou la taille des items
  autrebiblio:
    path: ...
#</bibliographies>
```

[1]

Par défaut, il y a toujours une bibliographie pour les livres. On peut définir son "tag" ici.

On pourrait avoir par exemple la bibliographie pour les personnages, dont on parlait plus haut, qui utiliserait dans le texte la balise personnage :

```
Il y a sur cette page personnage(Tom) qui intervient dans l'action. Il parle à personnage(Sarah), la pulpeuse sirène.
```

Elle sera définie ainsi dans le livre de recette du livre ou de la collection (avec un titre de troisième niveau, sur la page courante et les police, style et taille par défaut) :

```
#<bibliographies>
# ...
biblio:
# ...
personnages:
   title: "Liste des personnages"
   path: "ressources/personnages.yaml" [1]
   title_level: 3
   new_page: false
#</bibliographies>
```

[1]

Le fichier ./ressources/personnages.yaml (chemin relatif par rapport au dossier du livre ou de la collection) doit définir tous les personnages du livre.

Noter que l'affichage de cette bibliographie particulière pourra être <u>définie très précisément</u> par rapport aux informations qui seront données dans le fichier <u>personnages.yaml</u> qui contiendra peut-être la fonction du personnage, sa relation avec les autres personnages, ses caractères positives, négatives ou ambivalentes, etc.

[2]

Le tag doit toujours être au singulier.

[3]

On parle ici de l'affichage de la bibliographie à la fin du livre, si des items ont été trouvés.

table_of_content (données de table des matières)

(pour définir dans la recette du livre ou de la collection l'aspect de la table des matières)

```
# in recipe.yaml/collection recipe.yaml
#
table_of_content:
 title: "Table des matières"
 no title: false # cf. [1]
 title_level: null # 1 par défaut
 level max: 3 # niveau de titre maximum
 line height: 12 # hauteur de ligne
 lines before: 4 # nombre de lignes avant le premier item
 numeroter: true # pour numéroter cf. [3]
 separator: "." # caractère entre titre item et numéro
 add_to_numero_width: 0 # cf. [2]
 level1:
   indent: 0 # indentation des items de ce niveau
   font n style: null # "Police/style" pour ce niveau
   size: null # taille pour ce niveau
   numero size: null # taille de numéro pour ce niveau de titre
   separator: '.' # entre le titre et le numéro de page/paragraphe
 level2:
   indent: 10
   # etc.
 levelX: # cf. [4]
#
```

[1]

Si cette valeur est true, le titre "Table des matières" (ou autre) ne sera pas affiché. Cela peut servir à ne pas voir le titre, mais cela sert aussi lorsque l"on veut mettre un titre, mais que ce titre ne soit pas dans la table des matières elle-même. Dans ce cas, dans le fichier texte du livre, on met :

```
# {no-tdm}Table des matières
```

C'est le {no-tdm} qui fait que le titre "Table des matières" ne sera pas inscrit dans la table des matières elle-même.

[2]

Paramètre "maniaque" pour ajuster l'espace vide entre le dernier caractère de séparation et le numéro de page ou de paragraphe.

[3]

SI ce paramètre est à false, seuls les titres seront inscrits, sans numéro de page ou de paragraphe.

Tous les niveaux jusqu'à :level max doivent être définis.

inserted_pages (types de page à imprimer)

```
# in recipe.yaml ou collection_recipe.yaml
# La page créée au tout départ par Prawn (cf. [1])
book format:
 page:
    :skip page creation: true # (true par défaut)
#<inserted pages>
inserted pages:
 # La PAGE DE GARDE est une page vierge insérée juste avant
 # la page de titre
 page_de_garde: true # true par défaut
 # La PAGE DE TITRE est une page reprenant les informations
 # de la couverture ainsi que quelques informations supplémentaires
 page de titre: false # false par défaut
 # La PAGE DE FAUX TITRE est une page insérée avant la page de
 # titre et après la page de garde, et reprenant juste le titre
 # de l'ouvrage et son auteur.
  faux_titre: false # false par défaut
#</inserted_pages>
```

[1]

Au tout départ de la création d'un fichier PDF par Prawn est créé par défaut une page vierge. Pour empêcher ce comportement, afin de mieux maitriser la gestion des pages, il faut mettre ce paramètre à true (vrai)

Impression forcée des pages de type

Notez que certaines pages ne sont imprimées dans le livre que si les bornes correspondantes sont placées dans le livre. C'est le cas notamment de la table des matières, qui doit être stipulée par :

```
(( table_des_matieres ))
```

ou de l'index :

```
(( index ))
```

page_de_titre (définition de la page de titre)

La « page de titre » est la page qui se situe dans les premières pages du livre, reprenant le titre, l'autre, l'éditeur, etc.

```
# in recipe.yaml/collection recipe.yaml
#<page_de_titre>
page_de_titre:
 fonts:
   title: "Police/style"
                               # police pour le titre du livre
   subtitle "Police/style"
                                # police pour le sous-titre du livre
   author: "Police/style"
                               # police pour l'auteur
   publisher: "Police/style"
                               # police pour l'éditeur
   collection_title: null
                                # police pour le nom de la collection
  sizes:
   title: 18 # taille pour le titre du livre
   subtitle: 11 # taille pour le sous-titre
   author: 15 # taille pour l'auteur
   publisher: 12 # taille pour l'éditeur
   collection title: 12 # taille pour l'éditeur
  spaces before:
   title: 4  # nombre de lignes avant le titre
   subtitle: 1 # nombre de lignes avant le sous-titre
   author: 2  # nombre de lignes avant le nom de l'auteur
 logo: [1]
   height: 10 # Hauteur du logo
 paginate: false # [2]
#</page_de_titre>
```

[1]

Le chemin d'accès au logo est défini dans les informations sur la maison d'édition.

[2]

Par défaut, toutes les pages spéciales (titre, garde, faux-titre, etc.) ne sont pas numérotées (comme dans un « vrai » livre. Mais si pour une raison ou une autre, on veut les numéroter, il suffit de mettre cette propriété à true (pour chaque page à numéroter).

page_info (définition de la page des informations)

On appelle « page des informations » ou « page d'infos » la page, en fin de livre, qui indique les contributeurs à la fabrication du livre, depuis la maison d'édition et son éditeur jusqu'aux correcteurs, concepteurs de la couverture, ainsi que d'autres informations comme le numéro ISBN et la date de publication.

```
# in recipe.yaml/collection_recipe.yaml
#<page infos>
page_infos:
  aspect:
   libelle: # pour les libellés
      font_n_style: "Police/style"
      size: 10
      color: "CCCCCC"
   value: # pour les valeurs
      font_n_style: "Police/style"
      size: 10
    disposition: 'distribute' [2]
  # Données
  conception:
   patro: "Prénom NOM" # ou liste
           "prenom.nom@chez.lui" # ou liste [1]
 mise_en_page:
   # idem
 cover:
   # idem
 correction:
    # idem
 depot legal: "Trimestre ANNÉES"
 printing:
   name: "Imprimerie de l'Ouest"
   lieu: "Ours sur Orge"
  paginate: false # [4]
#</page_infos>
#<publishing> [3]
publishing:
          "<nom de la Maison d'éditions>"
 name:
  adresse: "numéro rue\nCode postal ville"
 mail:
           "<adresse mail principale>"
            "https://url.de.la.maison.fr"
  url:
#</publishing>
```

[1]

Cette liste doit être dans le même ordre que la liste :patro. Le premier mail sera attribué au premier patro, le deuxième mail au deuxième patro, etc.

[2]

Disposition générale dans la page. Les trois valeurs possibles sont :

'distribute'
Toutes les informations sont « distribuées » régulièrement sur la page 'top'
Toutes les informations sont placées au-dessus de la page 'bottom'
Toutes les informations sont placées en bas de la page

La page d'infos se sert aussi des informations de la maison d'édition.

[4]

Par défaut, toutes les pages spéciales (titre, garde, faux-titre, etc.) ne sont pas numérotées (comme dans un « vrai » livre. Mais si pour une raison ou une autre, on veut les numéroter, il suffit de mettre cette propriété à true (pour chaque page à numéroter).

page_index (données d"affichage de la page d'index)

Rappel: pour que la page d'index soit affichée dans le livre, il faut placer une balise ((index)) à l'endroit où on veut mettre cet index.

```
# in recipe.yaml/collection_recipe.html

#<page_index>
page_index:
    aspect:
        # Pour définir le MOT CANONIQUE
        canon:
            font_n_style: "Police/style" # pour le canon
            size: 10 # taille pour le canon
            # Pour définir l'aspect des nombres (pages ou paragraphes)
            number:
            font_n_style: "Police/style"
                 szie: 10

#</page_index>
```

Modules personnalisés (Expert)

[Expert] On peut créer des modules personnalisés qui vont permettre des traitements particuliers.

C'est nécessaire par exemple pour réinitialiser des valeurs lorsqu'un « deuxième tour » est nécessaire pour traiter toutes les références. Dans ce cas, il peut être nécessaire d'appeler une méthode de réinitialisation propre au livre, ou à la collection, pour remettre les compteurs à zéro. Dans le cas contraire, on prend le risque de se retrouver avec des doublons.

Cette méthode de réinitialisation s'appelle Prawn4book.reset.

On doit la définir dans un fichier **prawn4book.rb** à la racine de la collection ou du livre (si les deux fichiers existent, ils seront chargés).

Dans ce fichier, on écrit:

```
# in ./prawn4book.rb
module Prawn4book

def self.reset(first_turn) # [1]
  @is_first_turn = first_turn
```

```
# ... Tout ce qu'il faut faire au début de chaque tour
# ... d'impression du livre.
end

# Pour pouvoir faire n'importe où dans le programme :
# Prawn4book.first_turn?
def self.first_turn?
@is_first_turn === true
end
end
```

[1] +first_turn+ est à true lorsque c'est le premier tour et à false au deuxième tour (@rappel : un deuxième tour est exécuté lorsque des références arrières ont été trouvés et qu'il faut faire le livre en deux temps).

Dans ce fichier, on pourrait imaginer, par exemple, charger des parseurs et des formateurs si on ne veut pas utiliser les fichiers par défaut. Par exemple :

```
# in ./prawn4book.rb

require_relative 'assets/parseurs_formateurs/helpers'

module Prawn4book
end
```

Les scripts [experts]

Les *scripts* sont des petits programmes ruby qui permettent d'exécuter des opérations "externes" sur un livre ou toute une collection. Typiquement, on trouve par exemple, de base, un script qui permet de rogner les images SVG produites par une application externe comme *Affinity Publisher*. Typiquement, ils peuvent permettre par exemple d'ajouter un élément bibliographique ou d'obtenir le dernier ID utilisé pour une liste d'éléments.

Il y existe deux sortes de scripts :

- les scripts natifs de *Prawn-for-book* (ils se trouve dans ./lib/commandes/script/scripts/ et ne devraient pas être modifiés),
- les scripts propres aux livres et aux collections qui se trouvent dans un dossier ./scripts à la racine du dossier du livre ou de la collection.

Lancer un script

Pour lancer un script, on utilise la commande :

```
> pfb script [<nom ou partie du nom du script>[ <arguments>]]
```

Par exemple, pour rogner une image, on joue:

```
> pfb script rogner ./images/monplan.svg
```

Cette commande produira l'image ./images/monplan-rogned.svg qui se trouve dans le dossier images à la racine du livre.

Si on ne connait pas le nom du script, on peut simplement jouer la commande :

```
> pfb script
```

... qui va afficher la liste des scripts natifs et ceux définis pour l'application.

Créer un script

Un script étant chargé dans l'application, il bénéficie de toutes les fonctionnalités de l'application et notamment :

- le gem 'clir' (qui permet par exemple l'interactivité avec les méthodes Q.ask, Q.select, etc., qui permet les couleurs, la gestion des précédences etc),
- la connaissance du livre ou de la collection :

```
Prawn4book::Prawn4book.current # => instance du livre courant
Prawn4book::Prawn4book.current.recipe # => recette du livre courant
```

Arguments passés au script

Les arguments passés par la commande sont obtenus à partir de ARGV[1].

Par exemple, si on appelle:

```
> pfb script monscript 12 "Bonjour"
```

... alors on pourra utiliser:

```
# Dans ./scripts/monscript_qui_fait_quelque_chose.rb

douze = ARGV[1]
bonjour = ARGV[2]
```

Note: ARGV[0] contient le chemin d'accès au dossier du livre (ou de la collection) pour que le script puisse être utilisé même en dehors de *Prawn-for-book*.

Annexe

Sous-commandes

Note : toutes ces commandes doivent être exécutée depuis un Terminal ouvert dans le dossier du livre (pas de la collection).

Description	Commande	Note
Produire le PDF pour l'imprimerie	pfb build	
Obtenir de l'aide pour l'application	pfb -h	
Ouvrir le manuel	pfb manuel	Ajouter l'option –dev pour l'ouvrir en édition
Lister toutes les cibles présentent dans le texte	pfb targets	cibles peut être aussi utilisé
Choisir un itemp dans une bibliographie	pfb choose <biblio></biblio>	
Jouer un script (du livre, de la collection, ou commun)	pfb script <nom></nom>	
Lancer un assistant	pfb assistant	Affiche la liste des assistants

Exemples de codes

Exemple de méthode générale par paragraphe code

Imaginons qu'on ait besoin de mémoriser le numéro de paragraphe d'un endroit particulier du livre, sans rien faire d'autre.

On place dans le texte :

```
Un paragraphe.

(( memoriser_cette_ligne ))

Un autre paragraphe.
```

Puisque la méthode ne doit rien écrire, on la programme dans le module Prawn4book du fichier prawn4book.rb:

```
# in ./prawn4book.rb

module Prawn4book

def self.memoriser_cette_ligne(pdf) # [1]

@@ligne = pdf.current_cursor # ou autre valeur
end
end
```

[1] Puisqu'il y a un paramètre à la méthode et qu'aucun argument n'est transmis par le paragraphecode, le programme sait qu'il faut transmettre pdf.

Pour passer des arguments à une méthode générale, il faut absolument les écrire comme du « vrai » code. Par exemple, si c'est un string, il faut l'entourer de guillemets (dans Sublime Text, avec le thème, on les obtient en faisant ALT-MAJ-7):

```
(( methode_argument_string(mauvaise écriture) ))
# => erreur

(( methode_argument_string("bonne écriture") ))
# => OK
```

On passe plusieurs arguments normalement, ne les séparant par des virgules :

```
(( methode(:un_symbole, "un string", 12, {un: "hash"})
```

Style inline

Le *style inline* est un formatage qui utilise les balises HTML pour formater le texte. On peut le faire partout (c'est-à-dire dans tous les paragraphes et toutes les définitions de formatage, avec Prawn-for-book.

Pour rappel, les balises utilisables sont :

```
h
              gras
i
              italique
              souligné
u
strikethrough barré
sub
              subscript
sup
              exposant
font
        Définition d'une fonte, qui accepte :
        name
              Nom de la fonte (qui doit être définie)
                Taille du texte
        size
        character_spacing
                            Espacement entre les caractères
        Par exemple :
```

Snippets

Si vous utilisez le package sublime-text "Prawn-for-book", vous disposez des facilités suivantes :

Le signe \$ indique la position du pointeur (curseur).

Code tapé	Effet
</td <td><!-- \$--></td>	\$
	((\$))

Snippets personnalisés

Pour le moment, il n'existe pas de moyens simples et programmatique et créer des snippets. Mais si vous êtes à l'aise avec **Sublime Text**, vous pouvez en créer de nouveau dans le fichier

Conversion des dimensions

Une dimension string — par exemple "2cm" — peut-être convertie en points-scripts (points PDF) à l'aide de la méthode :

```
"2.3cm".to_f
"3mm".to_f
```

Mais le plus simple reste quand même d'utiliser :

```
2.3.cm
```

Grille de référence

La *grille de référence* est une "grille" abstraite (mais qu'on peut afficher) sur laquelle viennent s'inscrire toutes les lignes du texte du livre (qu'on appelle les **lignes de référence**). Dans un livre imprimé digne de ce nom, cette grille permet d'avoir les lignes alignées entre la page droite et la page gauche, mais aussi alignées par transparence, afin qu'une ligne d'une feuille précédente ou suivante n'apparaisse pas (trop).

Dans *Prawn-for-book* on règle cette grille de référence grâce au paramètres :line_height qui se définit dans le format du livre (ou de la collection).

On peut demander l'affichage de la grille de référence au moment de la conception du livre (par exemple pour compter le nombre de lignes à laisser entre deux éléments) en utilisant l'option :

```
pfb build -grid
```

Points PDF

Par défaut, les valeurs sont comprises en *points-PDF*. La valeur 12, par exemple, sera considérée comme "12 points-PDF".

Mais on peut tout à fait utiliser d'autres mesures en ajoutant l'unité après la valeur, séparée par un point (pas une espace). Par exemple :

```
12.mm # pour 12 millimètre
1.3.cm # pour 1 centimètre et 3 millimètre
# etc.
```

Les unités possibles sont : mm (millimètres), cm (centimètres), dm (décimètres), ft (unités impériales — anglaises), pt (points).

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.

Trouver le chemin vers une police

Sur Mac, pour obtenir facilement le chemin vers une police, on peut utiliser cette procédure :

- Ouvrir la fenêtre des polices dans n'importe quelle application (souvent en jouant # t,
- dans le petit menu général (en haut à gauche), choisir « Gérer les polices »,
 - => Cela ouvre l'application
- Ouvrir spotlight (*[Espace]) et taper « Li »,
- choisir d'ouvrir l'application « Livre des polices »,
- trouver la police voulue,

- ouvrir le menu contextuel en cliquant sur la police (^[clic sur police]),
- choisir l'item de menu « Afficher dans le Finder »
- dans le Finder, ouvrir le menu contextuel en cliquant sur le fichier de la police (^[clic sur fichier]),
- presser la touche vet choisir l'item de menu « Copier < nom de la police > en tant que nom de chemin »,
- le chemin d'accès au fichier de la police est placé dans le presse-papier, il suffit de le coller dans la recette du livre.

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 pfb.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 <code>.pfb.txt</code> (ou markdown), il faut éditer le fichier <code>Prawn4Book.sublime-settings</code> du package et choisir la <code>"font_face"</code> qui convient (en ajouter une si nécessaire). Régler aussi le <code>"font_size"</code> et <code>"line padding top"</code> pour obtenir le meilleur effet voulu pour un travail confortable sur le texte.

On peut ouvrir ce package dans Sublime Text à l'aide de :

\$> pfb open package-st

Modifier l'aspect du texte dans Sublime Text (son affichage dans l'application)

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

\$> pfb 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).

Prawn

Blocs de texte avec Prawn

[fichier helper.rb]: