Mailing

Permet d'envoyer le mailing d'un mail complexe à partir d'un simple fichier markdown (avec métadonnée).

En ligne de commande

```
send-mails <fichier md>[ <options>]
```

Options

Description	Options	Notes
Faire une simple simulation de l'envoi	`-s`	« s » pour « simulation ». En mode simulation, les délais entre les envois seront raccourcis et les mails seront enregistrés dans le dossier `tmp/mails` de l'application.
Ne pas laisser de délai entre les messages	`-d`	« d » pour « délai ». Sinon, un temps aléatoire sera laissé entre chaque message envoyé, pour donner l'impression d'envois réels.
Pour afficher l'aide	`-h`	« h » comme « help », aide en anglais
Ouvrir la version éditable du manuel	`-dev`	S'emploie donc avec `send-mail manuel -dev`

Fichiers markdown mailing

Ce fichier constitue l'élément principal de l'application ***Mailing***. C'est un fichier markdown (donc d'extension `.md`) qui définit précisément le mailing. Il est constitué de deux parties :

- les métadonnées,
- le message à transmettre.

Contenu des métadonnées

Les *métadonnées* sont consignées dans le fichier markdown du mailing, en haut, entre deux `---` :

```
---
<métadonnée>
---
<message>
```

Elles sont constituées de paires `Clé = Valeur` ou la clé commence toujours par une capitale suivie de minuscules.

Définitions obligatoires

```
Subject = Le sujet que prendront tous les mails
```

> Noter que les guillemets, quand c'est une simple valeur `String` qui est donnée, ne sont pas obligatoires.

Ce sujet peut contenir du code ruby dans `#{code}` qui sera évalué dans le contexte du destinataire ainsi que des variables destinataires `%{variable}` (par exemple `%{Patronyme}`. Voir Sujet dynamique.

```
From = <mail> # adresse mail de l'expéditeur
```

Cette donnée peut avoir la forme `Patronyme <mail valide>`.

```
To = <définition des destinataires>
```

On peut définir les destinataires de plusieurs façons. Voir la Définition des destinataires.

Définitions optionnelles

On peut donner au mailing un nom qui sera utile pour d'autres programmes :

```
Name = Le nom du mailing, juste pour information
```

On peut définir des exclusions avec la clé :

```
Excludes = <définition des exclusions>
```

On peut définir ces exclusions de plusieurs manières, de la même façon que les destinataires. Voir aussi Définition des exclusions.

Dans les métadonnées, on peut aussi définir **les images** qui seront utilisées dans le texte. Une image utilisée dans le texte possède toujours un identifiant de la forme `IMG<id>`. Par exemple `IMGlogo`. Dans l'entête, on doit absolument définir son chemin d'accès relatif ou absolu.

```
IMG<id image> = <path absolu ou relatif à l'image>
```

Par exemple, pour le logo qui se trouverait dans un dossier `images` au même niveau que le fichier mailing, on pourrait avoir :

```
---
...
IMGlogo = ./images/logo.jpg
---
Bonjour,

Voici mon logo :

IMGlogo
```

Contenu du message

Le contenu du message est la seconde partie du fichier mailing :

```
---
<métadonnées>
---
<définition des styles>

<message>
```

Voir Définition du message à envoyer.

Définitions

DESTINATAIRES

Les destinataires se définissent dans les [métadonnées][] à l'aide des propriétés `To` et `Exclude`. La propriété `To` permet de définir la liste de référence et la propriété `Exclude` permet de retirer des adresses.

Manière de définir destinataires

La propriété `To` peut définir (comme la propriété `Excludes` qu'on verra plus tard) :

1. Une simple adresse avec patronyme : `Patronyme <mail valide>` (dans une liste s'il y en a plusieurs).
2. Une définition minimale d'un destinataire sous forme de rangée CSV : `"<Patronyme>,<sexe>,<mail>"` (l'ordre n'importe pas). Voir la Définition sous forme rangée CSV.
3. Un chemin d'accès relatif ou absolu à un fichier CSV valide définissant les adresses et les propriétés des destinataires. C'est la forme qui permet le plus simplement de définir de nombreuses propriétés. Voir Définition par fichier CSV.
4. Une table de type ruby contenant toutes les données à commencer par `{'Mail' => "<mail>", 'Patronyme' => "<patronyme>", 'Sexe' => "<H/F>"}`. Voir Définition par table ruby.

5. Une liste contenant un ou plusieurs éléments précédents. Voir Définition dans une liste.

Définition des destinataires par rangée CSV

La façon la plus simple, qui ne correspond qu'à un seul destinataire est la suivante :

```
To = "Prénom NOM,H,mail@valide.com"
```

(noter que les guillemets sont optionnels)

Définition des destinataires par fichier CSV

Mais l'utilisation d'un mailing requiert plusieurs destinataires. Le plus simple est de les définir dans un fichier CSV bien formaté :

```
To = destinataires.csv
```

Avec cette formulation, le fichier `destinataires.csv` doit obligatoirement se situé au même niveau que le fichier mailing. Sinon, on peut indiquer le chemin relatif :

```
To = ../utils/destinataires.csv
```

... ou absolu :

```
To = /Users/phil/mails/destinataires.csv
```

On pourra trouver ci-dessous le formatage de ce fichier CSV.

Définition des destinataires par table ruby

On peut définir les destinataires par :

```
To = {'Mail' => "<mail>", 'Patronyme' => "<patronyme>", 'Sexe' => "<H/F>", 'Var1' =>
"...", 'Var2' => "...", etc. }
```

> Les clés doivent être obligatoirement des strings qui commencent par des capitales.

Pour avoir plusieurs destinataires, il suffit de mettre ces tables dans une liste :

```
To = [{'Mail' => "<mail>", 'Patronyme' => "<patronyme>", 'Sexe' => "<H/F>"}, {'Mail' =>
"<mail2>", 'Patronyme' => "<patronyme2>", 'Sexe' => "<H/F>"}, etc., ./fichier.csv]
```

> Noter, comme cela est précisé plus haut, qu'il est possible de mettre n'importe quelle donnée dans cette liste.

EXCLUSIONS

Les exclusions peuvent être définies de la même manière que les destinataires.

Exclusions par liste de mails

Le fichier possède le même formatage que le fichier CSV des destinataires.

La seule différence, pour les exclusions, est qu'on n'a besoin que de l'adresse mail. La définition des exclusions peut donc ressembler simplement à :

```
Excludes = ["<mail 1>", "<mail 2>", etc.]
```

Exclusions par filtrage

Une manière puissante d'exclure des destinataires (ou de les choisir précisément) consiste à utiliser un filtre en ruby (donc il est impératif de connaitre le langage ruby). On définit le fichier filtre dans la donnée `Excludes`, seul ou dans une liste :

```
---
...
Excludes = /path/to/filtre.rb
---
...
```

Ou :

```
---
...
Excludes = ["/path/to/exclusions.csv", "/path/to/filtre.rb"]
```

> Noter que ci-dessus, on procèdera d'abord à l'exclusions des mails contenus dans `exclusions.csv` et ensuite on enverra la liste restante à `filtre.rb`. L'ordre, dans `Excludes`, est capital.

> Grâce aux listes, on peut enchainer plusieurs filtres comme dans l'exemple en annexe.

Dans le fichier ruby, on définira une **méthode de classe qui portera le même nom que le fichier** et qui recevra en premier argument la liste actuelles des destinataires (déjà filtrés ou non).

Si on définit le mailing par :

```
---
To = desti.csv
Excludes = ./modules/filtre_femmes.rb
```

... alors on aura un fichier `filtre_femmes.rb` dans le dossier `modules` au même niveau que le fichier mailing qui contiendra :

```
# modules/filtre_femmes.rb
class Mailing

  def filtre_femmes(recipients)
    # Ici le traitement des recipients pour produire
    # la liste recipients_filtred qui sera retournée
    #
    return recipients_filtred
  end
end
```

Paramètres dynamiques

On peut transmettre à la méthode de filtrage d'autres arguments. Il suffit de les mettre à la suite du chemin d'accès au fichier, en indiquant leur type par leur forme (un nombre sera tel quel, une chaine sera mise entre guillemets, une table sera mise telle quelle, etc.).

```
---
...
Excludes = /path/to/mon_filtre.rb <arguments>
---
```

Ou dans une liste :

```
---
...
Excludes = ["out.csv", "filtre.rb <arguments>"]
---
```

On peut définir par exemple d'envoyer un nombre et une chaine avec :

```
---
...
Excludes = /path/to/mon_filtre.rb 25 "jeudi"
---
```

Ou dans une liste :

```
---
...
Excludes = ["out.csv", "/path/mon_filtre.rb 25 'jeudi'"]
---
```

> Noter les guillemets simples utilisés

On aura alors la définition de la méthode de filtre :

```
# mon_filtre.rb
class Mailing

  def mon_filtre(recipients, age, jour)
    recipients.reject do |rec|
      if rec.age > age
        false
      else
        # On ajoute "(du <jour>)" au patronyme (c'est idiot, juste
        # pour l'exemple)
        rec.data['Patronyme'] = rec.data['Patronyme'] + " (du #{jour})"
        true # le prendre
      end
    end
  end

end

# Ajout d'une méthode au destinataire
class Receiver
  def age
    Time.now.year - data['Naissance'].to_i
  end
end
```

> Noter comment nous ajoutons une méthode au destinataire (`class Receiver`) en bas de fichier

---

Définition dans une liste

Que ce soit les destinataires ou les exclusions, on peut toujours les définir dans une liste. Une liste est définie par des crochets qui doivent commencer et fermer la donnée complète :

```
To = [ <définition par liste> ]
Excludes = [ <définition par liste> ]
```

À l'intérieur de ces crochets, on peut trouver toutes les définitions possibles et mélangées, simples mails, table ruby, fichier CSV, rangée CSV, filtre ruby.

Le seul impératif est d'utiliser des guillemets sur toutes les chaines. Alors qu'il est possible de définir un destinataire unique sans guillemets, de la façon suivante :

```
To = Patro NYME, H, patronyme@chez.lui
```

… dans une liste, il faudra obligatoirement mettre des guillemets :

```
To = ["Patro NYME, H, patronyme@chez.lui", ...]
```

> Noter que c'est la donnée complète qui est entre guillemets, pas ses éléments. Cela permet de distinguer les éléments, qui seront ensuite décomposés.

Idem pour les fichiers :

```
To = /path/to/file.csv
```

Mais dans une liste :

```
To = ["/path/to/file.csv", ...]
```

Mais il ne faut pas mettre de guillemets autour de la définition par table ruby :

```
To = {'Mail' => "<mail>", 'Patronyme' => "<patronyme>", 'Sexe' => "<H/F>"}
```

Idem dans une liste :

```
To = [{'Mail' => "<mail>", 'Patronyme' => "<patronyme>", 'Sexe' => "<H/F>"}, ...]
```

> Il suffit, en fait, de se souvenir qu'une liste sera évaluée en ruby. Donc tout élément doit être « compréhensible » pour ruby.

Noter que `Excludes` peut être défini aussi par une liste, et que **l'ordre est alors capital**.

MESSAGE

Les styles

Le message, deuxième partie du fichier mailing, peut commencer par une définition des styles qui seront utilisés ensuite. Ces styles sont définis en haut du fichier et doivent tous commencer par un point suivi du nom du sélecteur (attention de ne pas confondre avec l'utilisation pure en CSS où les points sont utilisé pour les `class`. Ici, le point définit un sélecteur. Par exemple, si on a cette définition :

```
---
# Métadonnées
---
.mabalise {color: red;}
```

… alors on utilisera dans le message :

```
---
# Métadonnées
---
.mabalise {color: red;}

Bonjour,

Ce texte sera en <mabalise>rouge</mabalise.
```

... qui produira le message :

«
Bonjour,

Ce texte sera en rouge.
»

Le pseudo-format markdown

Ensuite, le message lui-même peut utiliser un marquage proche de markdown, mais simplifié. Il peut faire usage :

Description	Marquage	Notes
Italiques	`*...*`	
Gras	`**...**`	
Souligné	`_..._`	
Listes	`* <item>` `* <item>` `* <item>`	

Liste numérotée	`1. \<item>`
	`* <item>`
	`* <item>`
		Contrairement à markdown le premier item doit obligatoirement être `1.` et les autres items doivent être des astérisques.

Titres	`# titre`, `## titre`, etc.	Pour définir précisément leur style, définir `.h1`, `.h2`, `.h3` etc. en haut du fichier. Les points, avec les « h », sont obligatoires, contrairement à CSS.

Les images

On peut ensuite utiliser des images, en les définissant dans l'entête.

> Noter dès à présent qu'on peut définir très précisément la position et la taille d'une image en utilisant les styles comme c'est montré dans l'annexe.

Les variables

Les variables dans le message à envoyer peuvent être de deux natures :

- [Expert] variables ruby évaluées comme telles, une fois pour toutes quel que soit le destinataire,
- variables destinataire évaluées à la construction du mail pour le destinataire.

[Expert] Les **variables ruby** s'inscrivent comme en ruby à l'aide de `#{code à évaluer}`.

Les **variables destinataires** s'inscrivent à l'aide de `%{variable}`.

Variables communes

Parmi les variables générales, on trouve :

Description	Variable	Note
Patronyme du destinataire	`%{Patronyme}`	Noter la capitale, indispensable
Mail du destinataire	`%{Mail}`	
Prénom du destinataire	`%{Prenom}`	Si défini ou si le patronyme est défini
Nom du destinataire	`%{Nom}`	Si défini ou si le patronyme est défini
Toutes les variables féminines		
Madame/Monsieur	`%{Madame}`	
(rien)/e	`%{e}`	fort/forte
a/on	`%{a}`	mon / ma
elle/il	`%{elle}`	
et/ette	`%{ette}`	sujet/sujette
er/ère	`%{ere}`	première/première
ef/ève	`%{eve}`	bref/brève
f/ve	`%{ve}`	veuf/veuve
la/le	`%{la}`	
(rien)/ne	`%{ne}`	bon/bonne
eur/rice	`%{rice}`	correcteur/correctrice
x/se	`%{se}`	heureux/heureuse
ec/èche	`%{eche}`	sec/sèche
l/lle	`%{lle`	bel/belle
(rien)/sse	`%{sse}`	maitre/maitresse

Variables par colonne CSV

On peut ensuite avoir n'importe quelle variable qui serait définie dans le fichier CSV des destinataires. **La variable correspond alors au nom de la colonne**. Par exemple, si le fichier CSV des destinataires contient l'entête :

```
Id,Mail,Patronyme,Fonction,Annee
```

... alors on pourra utiliser dans le message (et dans le sujet) les variables `%{Fonction}` et `%{Annee}`.

Variables par méthode personnalisées [Expert]

[Expert] On peut ensuite définir des variables ruby (qui sont en fait des méthodes) dans le module propre au mailing. Il suffit de les définir comme des méthodes d'instances de la classe `Receiver`.

Par exemple :

```
class Receiver
  def ma_custom_variable
    # Retourne la valeur calculée pour le destinataire
  end
end
```

... permettra d'utiliser la variable `%{ma_custom_variable}` dans le message (ou le sujet).

Fichier CSV des destinataires

Un fichier destinataires bien formaté doit obligatoirement posséder une entête définissant les champs et des virgules pour séparer les données.

Le nom des champs doit être capitalisé et comprendre au minimum `Patronyme` (avec le nom en capitales), `Mail` et `Sexe` (de valeur `H` ou `F`). **L'ordre des champs n'importe pas**.

Les autres champs seront des données qu'on pourra reprendre à l'aide de variables `%{Variable}` dans le texte du message.

Donc, un fichier minimum ressemblera à :

```
Patronyme,Mail,Sexe
Prénom NOM,sonmail@chez.lui,H
Prénome NOM,autremail@chez.elle,F
```

Ce format vaut aussi bien pour les fichiers de destinataires que les fichiers d'exclusion.

Module propre au mailing [Expert]

[Expert] Les experts en ruby peuvent utiliser un module (donc ruby) pour gérer :

- les filtres,
- les variables personnalisées.

Le *module propre au mailing* est simplement un module ruby (donc un fichier `.rb`) qui porte le même nom (l'affixe, en fait) que le fichier markdown du mailing et se trouve au même niveau. Par exemple :

```
mon_dossier/autre_dossier/fichier_mailing.md
mon_dossier/autre_dossier/fichier_mailing.rb   # son module propre
```

Dans ce fichier, on pourra définir les variables par :

```
# in fichier_mailing.rb
class Receiver

  def ma_variable
  end

end
```

Pour le détail, voir Variables par méthode personnalisée.

Et on pourra définir les filtres par :

```
# in fichier_mailing.rb
class Mailing
  def une_methode_de_filtre(recipients)
    # traitement de recipients
    # ....
    return recipients
  end
```

Pour le détail, voir Exclusions par filtrage.

Annexes

Exemples de code

Sujet dynamique

```
---
Subject = Le sujet du #{Time.now.wday}
...
---
...
```

Le sujet dynamique peut aussi faire appel au destinataire en invoquant une de ses variables :

```
---
Subject = Bonjour %{Prenom}, nous sommes un #{Time.now.wday}
...
---
...
```

Mais notez bien, ci-dessus, l'utilisation du `#` (dièse) pour le code ruby (commun à tous les messages) et l'utilisation du `%` (signe pourcentage) pour faire appel à une propriété du destinataire en particulier.

[Expert] On peut également faire un traitement très précis en utilisant du code ruby qui utilise les données du destinataire courant en utilisant le fait que le code est évalué dans le contexte de l'instance `Receiver` du destinataire.

Imaginons par exemple que nous voulions enregistrer un message différent en fonction de la première lettre du nom du destinataire (pour créer trois groupes différents, de A à L, de M à T et de U à Z).

Le sujet final devra ressembler à : « **John, chanceux, vous êtes dans le 2e groupe** »

Dans le fichier mailing, on aura :

```
---
Subject = %{Prenom}, chanceu%{se}, vous êtes dans le %{indice_groupe} groupe
To = bons.csv
From = phil@chez.lui
---
Bonjour %{Patronyme}

Le titre précise le groupe dans lequel vous vous trouverez.
```

Si l'on lance ce mailing, il produit une erreur en précisant que la méthode `indice_groupe` est inconnue de la classe `Receiver`. Il nous faut préciser cette méthode.

Le fichier mailing s'appelant `mon_mailing.md`, le programme recherche un fichier s'appelant `mon_mailing.rb` au même niveau que le fichier mailing. Ce fichier contient :

```
# mon_mailing.rb
class Receiver

  # Cette méthode va retourner l'indice du groupe
  def indice_groupe
    premiere_lettre = nom[0]
    if premiere_lettre.match?(/[A-L]/)
      "1er"
    elsif premiere_lettre.match?(/[M-T]/)
      "2e"
    else
      "3e"
```

```
    end
  end

end
```

Cette fois, le mailing pourra être envoyé, avec le bon titre.

Images avec styles

Grâce aux styles, on peut définir très précisément la taille et la position des images dans le fichier. Par exemple, ci-dessous, on crée une rangée de vignettes de livres :

```
---
...
IMGlivre1 = ./images/livre1.jpg
IMGlivre2 = ./images/livre2.jpg
IMGlivre3 = ./images/livre3.jpg
---
.vignette {display: inline-block; widht:120px;margin:10px;}

Bonjour,

Voici nos derniers livres :

<vignette>IMGlivre1</vignette><vignette>IMGlivre2</vignette>
<vignette>IMGlivre3</vignette>

Bien à vous.
```

Cela produit le mail suivant, avec les trois images sur la même ligne, de taille 120 pixels en largeur et séparés de 20 pixels.

> Noter que les trois vignettes doivent s'inscrire impérativement sur la même ligne (sans retour chariot). Et si possible, sans espace entre les balises.

Chaine de filtres

Grâce aux listes, on peut enchainer plusieurs listes. Par exemple :

```
---
...
To = destinataires.csv
Excludes = ["filtre_hommes.rb", "filtre_age.rb 25"]
---
Bonjour %{Patronyme},

Vous êtes une femme âgée de plus de 25 ans.
```

Le mailing ci-dessus utilise la liste des destinataires `destinataires.csv` qui définit les propriétés normales ainsi que la propriété `Naissance` de l'année de naissance :

```
Id,Patronyme,Mail,Sexe,Naissance
1,Lui NOM,sonmail@chez.lui,H,2000
2,Elle NOM,sonadresse@chez.elle,F,2003
etc.
```

Le premier filtre va exclure tous les hommes, il est défini ainsi :

```
# filtre_hommes.rb
class Mailing
  def self.filtre_hommes(destinataires)
    destinataires.reject do |r|
      r.sexe == 'H'
    end
  end
end
```

Le second filtre va ne prendre que les destinataires restants de plus de 25 ans. Noter que cette méthode prend en second argument le nombre 25.

```
# filtre_age.rb
class Mailing
  def self.filtre_age(destinataires, age_minimum)
    # On définit l'année courante pour pouvoir calculer
    # l'âge
    current_year = Time.new.year
    # On boucle sur les destinataires pour les filtrer
    destinataires.reject do |r|
      # On calcule l'âge du destinataires
      age = current_year - r.data['Naissance'].to_i
      age < age_minimum
    end
  end
end
```