

# Mail-Manager — Gestion des mails

---

## Description générale

---

**MailManager** est une commande terminale ( `send-mail` ) et un **gem** qui permet de fonctionner à trois niveaux :

1. l'envoi de simple texte, à une personne en particulier,
2. l'envoi d'un mail modèle à un ensemble d'adresses défini dans des fichiers
3. l'envoi de mail-type à une ou plusieurs personnes
4. [utilisation comme API](#) en transmettant le message `string`, la liste des instances destinataires et optionnellement (sic) des options.

## Mail/mailing en ligne de commande

Jouer la commande :

```
send-mail path/to/mail.md[ <options>]
```

Le fichier `path/to/mail.md` qui définit toutes les données doit être [correctement formaté](#).

On trouve en annexes toutes les [options de la ligne de commande](#).

## Fichiers requis

Pour fonctionner, l'app s'appuie sur :

- un [fichier markdown obligatoire](#) qui définit aussi bien le message que les destinataires, l'expéditeur, le sujet, etc.
- un [module ruby](#) (obligatoire pour les mails type qui permet de lier des opérations quelconques à l'envoi d'un mail (à commencer par son archivage)

---

## Fichiers

### Fichier message

Son format global est :

```
---  
<metadata>  
---  
  
<message markdown>
```

# Fichier module

Il doit obligatoirement porter le même nom (affiche) que le message, avec bien sûr l'extension ruby `.rb`.

Il peut contenir :

```
module MessageExtension

  # Méthodes qui étendent la class MailManager::Message
  # Les variables propres dans le mail sont définies ici.

end

module MessageClassExtension
  # Idem pour les méthodes de class
end

module SourceFileExtension

  # Méthodes qui étendent la class MailManager::SourceFile

end

module SourceFileClassExtension
  # Idem pour les méthodes de class
end

module RecipientExtension

  # Méthodes qui étendent la class MailManager::Recipient

end

module RecipientClassExtension
  # Idem pour les méthodes de class
end

module SenderExtension

  # Méthodes qui étendent la class MailManager::Sender

end

module SenderClassExtension
  # Idem pour les méthodes de class
end
```

---

# Définition du mail

---

## NOM DU MESSAGE

Tout fichier markdown définissant un mail (de mailing ou mail-type) peut commencer par définir son nom (`Name`) qui servira autant à le décrire qu'à en parler dans les messages.

```
---
Name = "Fichier type envoyé suite à l'envoi de l'exemplaire découverte"
Type = mail-type
...
---
```

Note : si cette valeur n'est pas fourni, c'est le nom du fichier, simplifié, qui sera utilisé.

---

## DESTINATAIRE(s)

### Définition du ou des destinataires

On peut définir un ou plusieurs destinataires, par fichier ou par valeur explicite. Ces destinataires se définissent grâce à la métadonnée `To` de la manière suivante.

Par valeur explicite :

```
---
To = philippe.perret@yahoo.fr
---
```

Avec un patronyme :

```
---
To = Phil <philippe.perret@yahoo.fr>
---
```

Par valeur explicite avec plusieurs destinataires :

```
---
To = [ "mail1@chez.lui", "mail2@chez.eux", "Phil <mailphil@chez.lui" ]
---
```

Avec des valeurs explicites, un sexe et un patronyme précisés :

Note : l'ordre importe peu, l'application est capable de reconnaître le type de la donnée.

```
---  
To = ["H,Patrick,patrick@gmail.com"]  
---
```

OU :

```
---  
To = "H,Patrick,patrick@gemal.com"  
---
```

Par liste d'adresses :

```
---  
To = /path/to/liste/adresses.csv  
---
```

Par méthode de classe :

```
---  
To = :ma_methode_de_class  
...  
---
```

Cette méthode doit être défini dans le [fichier module du mail](#), en tant que **méthode de classe**, donc dans un module `RecipientClassExtension` :

```
# in <affixe-mail>.rb  
module RecipientClassExtension  
  
  def ma_methode_de_class  
    # ...  
    # @return liste des instances de destinataires  
  end  
  
end #/module
```

## Liste d'adresses dans fichier

Pour fonctionner avec **MailManager**, un fichier contenant une liste d'adresses doit respecter certaines règles :

- Si c'est un fichier `YAML`, ça doit être une liste (`Array`) d'éléments qui définissent tous, au minimum, la propriété `:mail` (ou `'mail'` ou `'Mail'` et la propriété `:sexe` définissant le sexe du destinataire, par `F` ou `H`.

- Si c'est un fichier `csv`, il doit impérativement :
    - utiliser la **virgule** comme délimiteur de données,
    - posséder une entête avec le **nom des colonnes**,
    - définir la colonne `Mail` et la colonne `Sexe` (valeur `H` ou `F`),
    - il peut définir la colonne `Patronyme` avec la patronyme de la personne,
    - il peut définir la colonne `Fonction` définissant la fonction du destinataire.
- 

## Définition des exclusions

Les « exclusions » correspondent aux emails à qui on ne doit pas envoyer les messages dans une liste de destinataires ([définie par fichier par exemple](#)).

On les définit en définissant la propriété `Excludes` dans les [métadonnées](#).

Par exemple :

```
---  
# ...  
Excludes = path/to/file.csv  
---
```

ou :

```
---  
# ...  
Excludes = "monadresse@chez.moi"
```

ou :

```
---  
# ...  
Excludes = ["Patrick <patrick@chez.lui>", "Moi,F,marion@chez.elle"]  
---
```

## TEXTE DU MAIL

C'est un texte au format markdown, donc utilisant des marques de formatage simples comme l'étoile pour les italiques ou la double étoile pour le gras. Les titres sont précédés par des dièses.

## Définition des variables

Il existe deux types de variable : les variables qui dépendent des destinataires (quand c'est un mailing-list par exemple) et les variables qui permettent de simplifier le code (typiquement : pour les images).

## Définition des variables template

Les “variables-template” sont définies une fois pour toutes dans le message à envoyer. Elles fonctionnent de façon très simples, avec un identifiant (en général majuscule pour le repérer plus facilement) dans le texte et sa définition dans le corps du message. Par exemple :

```
---
...
SRPS = "Savoir rédiger et présenter son scénario"
---

Cher ami,

Avez-vous lu « SRPS » ? Si ce n'est pas le cas, je vous conseille
de l'acheter car « SRPS » est un livre intéressant pour la rédaction
du scénario.
```

Noter, ici, l'utilisation d'aucun signe permettant de reconnaître la variable dans le message. Ceci pour y gagner au niveau de la lisibilité.

## Insérer une image

Les *variables-templates* permettent d'insérer de façon simple une image (en dur) dans le code consiste à utiliser une variable qui :

- commence par `IMG`,
- définit le chemin d'accès au fichier image.

Par exemple :

```
---
from = ...
to = ...
IMG1 = /path/to/mon/image.jpg
IMG1-alt = Son nom par défaut de l'image...
---

Bonjour,

Que penses-tu de cette image ?

IMG1

Cool, non ?
```

Ci-dessus, la variable `IMG1` sera remplacée par le code en dur de l'image de path `/path/to/mon/image.jpg`.

La variable `IMG1-alt` permet de définir la légende par défaut mais n'est pas obligatoire.

## Insérer une table

On le fait comme dans kramdown, par exemple :

```
| premier | deuxième | troisième |
```

Par défaut, la table prendra toute la largeur de la “feuille” et chaque colonne aura une largeur égale, définie en fonction du nombre de colonnes.

On peut définir l’alignement des contenus avec :

```
| premier | center::deuxième | right::troisième |
```

Ci-dessus, le mot “deuxième” sera centré et le mot “troisième” sera aligné à droite.

## Définition des variables destinataire

Dans le message, elles sont repérées par la code template `%{nom}`. Par exemple :

```
---
To = ...
From = ...
Subject = ...
---
Cher %{patronyme},

Allez-vous mieux ?
```

**Le nom de la variable est obligatoirement en minuscule**, même si elle est définie en majuscule dans le fichier de données.

Dans le code, on peut utiliser les variables classiques (`mail`, `patronyme`, `fonction`) mais on peut aussi utiliser n’importe quelle propriété qui serait définie dans le [fichier module](#), en tant que méthode du module `RecipientExtension` puisqu’il s’agit toujours de propriétés propres aux destinataires. Par exemple :

```
# in <affixe mail>.rb
module RecipientExtension

  def album
    @album ||= begin
      self.books.last.title # valeur utilisée
    end
  end
end
```

**Attention** : assurez-vous toujours que cette donnée soit définie pour tous les destinataires.

Par exemple, si tous les destinataires définissent la propriété `album`, on peut avoir :

```
---
To = ...
From = ...
Subject = ...
---
Cher %{patronyme},

Avez-vous terminé de lire %{album} ?
```

## Messages sexués

On peut utiliser une sexualisation du message (différents suivant femme ou homme, quand la propriété `sexe` est définie) grâce aux *propriétés féminines* (ou "féminines"). Par exemple :

```
---
...
---
Ch{%ere} ami{%e},

Vous êtes trop bon{%ne} avec moi. Si vous n'êtes pas {%la} destinataire de ce message,
tant pis, je ne vous aurez pas oublié{%e}.

etc.
```

Note implémentation : ces propriétés sont définies dans la constantes `FEMININES` dans le fichier `constants.rb` dans le cas où il faille en ajouter.

## Traitement après envoi

Grâce au [module qui accompagne le mail](#), on peut faire un traitement particulier après l'envoi du fichier. Typiquement, ce traitement peut ajouter une ligne à un historique qui garde la trace des envois.

Cette méthode s'appelle `:after_sending` et c'est une méthode d'instance de `MailManager::Sender`. Elle doit donc être définie dans le module `SenderExtension` et reçoit comme argument : le destinataire (instance `MailManager::Recipient`) et la fichier source (instance `MailManager::SourceFile`).



```
# in <affiche-message>.rb

module SenderExtension

  def after_sending(recipient, srcfile)
    #
    # Cette méthode est appelée après chaque envoi réussi, avec
    # l'instance MailManager::Recipient du destinataire et
    # l'instance MailManager::SourceFile du fichier message
    #
  end
end
```

---

## Envoi de mail-type

---


Un *mail type* est un mail dont le contenu peut varier en fonction du contexte. Typiquement, il a été mis en place lorsqu'il fallait confirmer l'envoi des exemplaires découvertes d'analyse aux conservatoires.

Prenons cet exemple pour comprendre concrètement comment cela marche.

Ce mail contenait, en variables :

- le nom du destinataire ("Monsieur untel"),
- le livre qui avait été envoyé,
- la date exactement de réception du livre

Son contenu final devrait être quelque chose comme :

En sujet :  ICARE éditions : Votre exemplaire découverte

En message :

Bonjour monsieur Untel Dutel,

Ce message pour vous informer que votre exemplaire gratuit de « Comprendre et apprendre le Premier prélude de Bach » vient de vous être expédié.

Sauf incident, ce livre devrait vous parvenir le lundi 27 mars prochain.

En vous remerciant de votre intérêt et vous en souhaitant bonne lecture,

Bien à vous,

Les Éditions ICARE

-----  
[Logo]

```
https://icare-editions.fr
```

Ce message est défini par :

```
---
Type = mail-type
Subject = 🎵 ICARE éditions : Votre exemplaire découvre de #{livre.titre_court}
From = administration@icare-editions.fr
To = /path/to/adresse/conservatoires.csv
# Pour ne pas proposer ceux qui l'ont déjà reçu
Excludes = /path/to/conservatoires_clients.csv
# Pour le logo
IMGlogo = /path/to/image/logo
# Pour savoir comment traiter les données
Data = module_mail_type.rb
---

Bonjour #{madame} #{patronyme},

Ce message pour vous informer que votre exemplaire gratuit de « #{livre.titre} » vient
de vous être expédié.

Sauf incident, ce livre devrait vous parvenir le #{jour_date} prochain.

En vous remerciant de votre intérêt et vous en souhaitant bonne lecture,

Bien à vous,

Les Éditions ICARE<br />
-----
IMGlogo <br />
https://icare-editions.fr
```

Remarquez les code `#{...}`. Il doivent pouvoir être définis par le module `module_mail_type.rb` défini dans `Data =` dans les métadonnées.

La valeur doit être un chemin absolu ou le nom du module, qui doit alors obligatoirement se trouver au même niveau que le message du mail-type.

Ce fichier implémente le module `MailTypeModule` qui doit définir les propriétés-méthodes utilisées par le mail-type. On trouve par exemple ici :

Ce code est volontairement complexe pour montrer les possibilités infinies

```
module MailTypeModule

#
# La liste des livres concernés par ce mail-type
#
Livre = Struct.new(:titre, :titre_court)
```

```

CHOIX_LIVRE = [
  Livre.new("Comprendre & apprendre le premier prélude en Do de BACH", "Prélude de
BACH"),
  Livre.new("Comprendre & apprendre le clair de lune de BEETHOVEN", "Clair de lune"),
  Livre.new("Comprendre & apprendre Gens et pays lointains de R. SCHUMANN", "Pays
lointains"),
  Livre.new("Gammes et accords dans tous les tons", "Gammes et accords"),
].map do |book|
  {name: book.titre_court, value: book}
end

def livre
  @livre ||= begin
    ### C'est ici que l'application demande le livre pour ###
    ### pouvoir écrire livre.titre et livre.titre_court ###
    clear
    Q.select("Pour quel livre ?".jaune, CHOIX_LIVRE, **{per_page:CHOIX_LIVRE.count})
  end
end

def jour_date
  ### C'est ici que l'application demande la date de réception ###
  ### qui est définie par '#{jour_date}' dans le code du mail ###
  now = Time.now + 7.jours
  auj = [['lundi', 'mardi', 'mercredi', 'jeudi', 'vendredi', 'samedi', 'dimanche'][now.wday -
1]]
  auj << now.day
  auj << MOIS[now.month][:long]
  auj = auj.join(' ')
  Q.ask("Date de réception (p.e. 'mardi 15 août')".jaune, **{default:auj})
end

end #/module MailTypeModule

```

Bien sûr, on pourrait imaginer un code beaucoup plus simple, comme :

```

---
# ...
Data = fichier_module.rb
---
Bonjour,

Nous sommes le #{jour_humain}.

```

Avec un module, au même niveau que le mail-type :

```
# Dans fichier_module.rb

module MailTypeModule

  def jour_humain
    Time.now('%d %m %Y')
  end

end

end
```

---

## Utilisation dans une application ruby

### Prise en main rapide

```
require 'mail_manager'

MailManager.send('/Users/phil/lemail.md')
```

Avec le fichier au chemin `path_mail_file` qui contient :

```
---
from = philippe.perret@yahoo.fr
to =   phil@atelier-icare.net
---
Bonjour à toi, Phil,

Comment ça va ?

Phil
```

---

## API

*MailManager* peut être utilisé comme API, grâce à la méthode `MailManager::API.send` :

```
require 'mail_manager'

retour = MailManager::API.send(message, destinataires, params)

# @param [String] message      Le message à envoyer, en String, avec les
#                               variables destinataire
# @param [Array] destinataires Liste des instances destintaires. Voir ci-
#                               dessous les méthodes requises
# @param [Hash] params         Options, notamment pour savoir si c'est une
#                               simulation ou pas.
```

## Message pour l'API

Ici, c'est un simple texte qui peut ne contenir que le texte lui-même

## Destinataires pour l'API

C'est une liste d'instances de classe quelconque avec pour seul impératif de répondre aux méthodes suivantes :

```
# mail                Retourne l'adresse mail seule
# patronyme           Le patronyme
# femme?              Retourne true si c'est une femme
# homme?              Retourne true si c'est un homme
# variables_template  Reçoit en premier argument la liste Array des variables contenues
#                    dans le message. Cette méthode devrait donc retourner une table
#                    qui
#                    définit toutes les clés, comme ci-dessous

def variables_template(ary)
  {
    ary[0] => "#{mail}",
    ary[1] => patronyme,
    une_autre => "pour voir",
    etc.
  }
end
```

## Options pour l'API

C'est table doit contenir au moins le titre ( `subject` ) du message à transmettre.

```
params = {
  subject:      "Sujet du message",
  sender:       'patronyme<mail@chez.lui>',
  ### Optionnel ###
  simulation:   true, # true => simuler l'envoi,
  no_delay:     true, # true => aucun délai entre les envois (pas
                    # recommandé, sauf pour les simulations)
  name:         'Nom de l'envoi',      # juste pour le suivi
}
```

---

## Définition de la police et de la taille

Utiliser `font_family = ...` et `font_size = 14pt` dans les métadonnées.

Par défaut, la police est 'Times' et la taille est '14pt'.

---

## Annexe

### Options de la ligne de commande

<code>-s/--simulation</code>	Pour faire simplement une simulation d'envoi
<code>-t/--test</code>	Pour faire un envoi seulement à des destinataires test (qui vont pouvoir vérifier l'aspect du message. Ils doivent être définis, pour le moment, dans <code>TEST_RECIPIENTS</code> dans <code>constants.rb</code> )
<code>-a/--admin</code>	Pour envoyer seulement à la personne définie comme l'administrateur dans <code>constants.rb</code> ( <code>ADMINISTRATOR</code> )
<code>-e/--mail_errors</code>	Pour re-procéder au dernier envoi en utilisant les mails qui ont échoué lors de ce dernier envoi (les mails ont été mis de côté et le problème doit avoir été résolu).
<code>-d/--no_delay</code>	Pour ne pas temporiser les envois (1 seconde entre simplement)