# Les #MercrediFiction de Mastodon, techniques pour les traduire dans d'autres formats

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 9 juillet 2017

https://www.bortzmeyer.org/mastodon-mercredifiction.html

---

Une des pratiques les plus intéressantes de Mastodon est celle du #MercrediFiction. Le mercredi, des gens écrivent une fiction, a priori en moins de 500 caractères. Plusieurs personnes avaient demandé si on pouvait publier ces pouètes sous d'autres formats (la demande initiale était pour créer un « "*ebook*" »). J'ai donc fait un petit programme qui publie les pouètes <https://mercredifiction.bortzmeyer.org/> #MercrediFiction en EPUB et HTML. Cet article, destiné à des programmeurs, explique les choix techniques et quelques problèmes rencontrés. Si vous n'êtes pas programmeur, n'hésitez pas à lire #MercrediFiction quand même !

Même moi, je m'y suis mis <https://mastodon.gougere.fr/@bortzmeyer/603740> à en écrire. Et pour les récupérer ? Premier choix, j'ai décidé d'écrire le code en Python. J'ai utilisé la version 3 de Python (qui n'est pas compatible avec la précédente), la version 2 n'évoluant plus.

Comment récupérer les pouètes ayant le bon mot-croisillon ? La solution évidente est d'utiliser l'API de Mastodon, qui est bien documentée <https://github.com/tootsuite/documentation/blob/master/Using-the-API/API.md>. Mais je la trouve trop compliquée pour mon goût, surtout question authentification (même si ce projet particulier n'a pas besoin d'authentification). J'ai donc décidé de déléguer cette tâche à l'excellent outil en ligne de commande madonctl <https://github.com/McKael/madonctl>. (On aurait pu utiliser toot <https://carlchenet.com/automatically-send-toots-to-th>.) Récupérer les pouètes ayant le mot-croisillon #MercrediFiction et les exporter au format JSON se fait simplement avec :

```
% madonctl --output json timeline :MercrediFiction
```

Je ne sais pas exactement jusqu'où madonctl, ou plutôt l'API de Mastodon qu'il utilise, remonte dans le temps. Je fais donc plusieurs récupérations par jour, et je supprime ensuite les doublons.

Quelle instance utiliser pour récupérer les pouètes ? Dans une fédération, rien ne garantit que deux instances (deux nœuds Mastodon) ont le même contenu. L'algorithme exact qui détermine dans quel cas une instance a reçu un pouète n'est pas, à ma connaissance, documenté (sauf dans le code source, feront remarquer les libristes). Il semble que ce soit à peu près « on récupère les pouètes des gens qui sont sur mon instance, ainsi que ceux repouètés par quelqu'un de mon instance, ainsi que ceux émis par quelqu'un suivi par quelqu'un de mon instance ». Dans ce cas, il est clairement évident qu'il vaut mieux récupérer les pouètes sur une grosse instance, avec plein d'utilisateurs et plein de connexions avec d'autres instances. J'ai choisi Mamot <https://mamot.fr/about>, qui a ces caractéristiques, et est en plus géré par des gens bien.

Une fois le compte mercredifiction@mamot.fr créé, l'appel à madonctl me donne donc les pouètes en JSON. Qu'en faire à la fin de la journée ? (Au passage, le code est évidemment distribué sous une licence libre, et il est visible en .)

Les deux formats que je crée actuellement, EPUB et HTML, utilisent tous les deux XHTML. Il faut donc traduire le JSON en XHTML, ce qui est trivial. J'utilise la bibliothèque officielle de Python pour analyser le JSON et, pour produire le XHTML, le module Yattag <http://www.yattag.org/>, dont le gros intérêt est que le gabarit est lui-même en Python : on n'utilise donc qu'un seul langage. Voici un exemple du code Python+Yattag utilisé pour produire le XHTML :

```
with tag('head'):
    with tag('link', rel = "stylesheet", type = "text/css", href = "mercredifiction.css"):
        pass
    with tag('title'):
        text("Mercredi Fiction %s" % datestr)
with tag('body'):
    with tag('h1'):
        text("Mercredi Fiction du %s" % formatdate(datestr, short=True))
```

Et le contenu du pouète lui-même ? L'API le distribue en HTML. Mais il y a deux pièges : c'est du HTML et pas du XHTML, et surtout rien ne garantit que l'instance n'enverra que du HTML sûr. Il y a un risque d'injection si on copie ce HTML aveuglément dans le document EPUB ou dans une page Web. Si demain l'instance envoie du JavaScript, on sera bien embêté.

Donc, première chose, traduire le HTML en XHTML. C'est simple, avec le module ElementTree (etree) de lxml <http://lxml.de/> :

```
from lxml import etree, html
...
tree = html.fromstring(content)
result = etree.tostring(tree, pretty_print=False, method="xml")
doc.asis(result.decode('UTF-8'))
```

(La fonction asis vient de Yattag et indique de mettre le contenu tel quel dans le document HTML, sans, par exemple, convertir les > en >.)

Et pour la sécurité ? On utilise un autre module de lxml <http://lxml.de/>, Cleaner :

```
from lxml.html.clean import Cleaner
...
cleaner = Cleaner(scripts=True, javascript=True, embedded=True, meta=True, page_structure=True,
                                    links=False, remove_unknown_tags=True,
                                    style=False)
```

Le code ci-dessus supprime de l'HTML le JavaScript, et plusieurs autres éléments HTML dangereux.

Voilà, on a désormais du XHTML propre et, espérons-le, sûr. Pour produire les pages Web, ça suffit. Mais, pour l'EPUB, c'est un peu plus compliqué. À ma connaissance, le format EPUB ne bénéficie pas d'une norme publique. J'ai donc dû me rabattre sur d'excellentes documentations comme le "*Epub Format Construction Guide*" <http://www.hxa.name/articles/content/epub-guide_hxa7241_2007.html> qui explique bien la structure d'un fichier EPUB. Un fichier EPUB est un zip contenant entre autre des fichiers HTML. Le lecteur EPUB typique est bien plus strict que le navigateur Web typique, et il faut donc faire attention à l'HTML qu'on génère. Il faut aussi placer diverses métadonnées, en Dublin Core. En l'absence d'une norme disponible, certaines choses doivent être devinées (par exemple, mettre le fichier CSS dans l'EPUB et le manifeste ne suffit pas, il faut aussi le déclarer dans l'HTML, ou bien le fait que les compteurs du nombre de pages dans la table des matières semblent optionnels). Pour les programmeurs Python, il y a aussi cet exemple en Python <https://gist.github.com/anqxyr/6c70a2e4e8209cd43fc1>.

Si vous écrivez un programme qui produit de l'EPUB, n'oubliez pas de valider l'EPUB produit. Cela peut se faire en ligne de commande avec le programme epubcheck :

```
% epubcheck mercredifiction-dernier.epub
Validating using EPUB version 2.0.1 rules.
No errors or warnings detected.
epubcheck completed
```

Ou cela peut être vérifié en ligne <http://validator.idpf.org/>, ce service utilisant le même programme.

Enfin, question EPUB, n'oubliez pas de servir les fichiers EPUB avec le bon type MIME, `application/epub+zip`, pour que le lecteur d'"*ebooks*" soit lancé automatiquement. Sur Apache, ça peut se faire avec la directive `AddType application/epub+zip .epub`.

Quelques petits détails amusants ou énervants, pour terminer. D'abord, l'API de Mastodon, pour des raisons qui m'échappent complètement (j'ai entendu des explications, mais toutes mauvaises), lorsque le pouète contient des emojis (comme [Caractère Unicode non montré [1] ] ou [Caractère Unicode non montré ]), renvoie non pas le caractère Unicode correspondant, mais du texte entre deux-points, par exemple `:cherry_blossom:` (ces textes viennent du système commercial Emoji One <https://www.emojione.com/>). Il n'y a absolument aucune raison d'utiliser ces séquences de texte. La bogue a été signalée <https://github.com/tootsuite/mastodon/issues/717> mais pour l'instant sans résultat. J'avais envisagé de faire la conversion moi-même <https://framagit.org/bortzmeyer/MercrediFiction/issues/2> pour #MercrediFiction mais c'est trop complexe, vu la taille de la base des emojis, à moins d'installer le code très compliqué fourni par Emoji One.

1. Car trop difficile à faire afficher par LaTeX

Autre limite de la version actuelle de mon service, le fait que les images ne sont pas incluses <https://framagit.org/bortzmeyer/MercrediFiction/issues/3>. Rare sont les pouètes #MercrediFiction qui comportent des images, mais ce serait tout de même sympathique de les gérer proprement. L'EPUB et le HTML devraient sans doute être gérés différemment : dans le premier cas, le lecteur s'attend à un "*ebook*" autonome, fonctionnant sans connexion Internet, et il faut donc récupérer les images en local, et les mettre dans le fichier EPUB. Dans le second, il vaut sans doute mieux mettre un lien vers l'image originale (avec toutefois le risque que, si l'hébergement original disparait, on perde l'image).

Notez que récupérer les images présente quelques risques juridiques. En théorie, c'est pareil avec les textes (le droit d'auteur s'applique aux deux, et je dois avouer que je récolte les pouètes sans penser à ce problème) mais, en pratique, le risque de réclamation est plus élevé pour les images (demandez à n'importe quel hébergeur). Pour l'instant, je n'ai pas encore reçu de message d'un juriste me demandant de supprimer tel ou tel pouète.

Le résultat est visible ici <https://mercredifiction.bortzmeyer.org/>. Sinon, il existe un autre service de distribution <http://mercredifiction.xyz/>, dont les sources (également en Python) sont disponibles <https://github.com/Meewan/MercrediFiction>.