RFC 7764 : Guidance on Markdown: Design Philosophies, Stability Strategies, and Select Registrations

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 23 mars 2016

Date de publication du RFC : Mars 2016

https://www.bortzmeyer.org/7764.html

Ce RFC est un complément du RFC 7763 [1], qui enregistrait le type MIME `text/markdown`. Il contient tout ce qui ne tenait pas dans le RFC original, c'est donc un pot-pourri de considérations philosophiques, de descriptions des variantes les plus courantes de Markdown, de techniques de stockage local de contenus en Markdown...

Le RFC 7763 était plus sec, contenant uniquement ce qui était nécessaire à l'enregistrement de `text/markdown`. Cet autre RFC est plus disert, expliquant d'abord (section 1) la philosophie de Markdown, et ses usages. Rappelons donc que Markdown est un format de marquage du texte. Lorsqu'on veut stocker des textes (romans, rapports techniques, articles scientifiques, lettres d'amour, etc) sur un système informatique, on a plusieurs solutions : elles vont du texte brut à un format binaire. Le texte brut est le stockage des seuls caractères Unicode (cf. RFC 6838, section 4.2.1 : le texte est distribué sur l'Internet avec un type MIME `text/quelquechose`, par exemple `text/plain` pour le texte brut). Son gros avantage est qu'il est lisible et modifiable avec n'importe quel logiciel. Bon, en fait, le texte brut n'est jamais tout à fait brut. Il contient quelques caractères de contrôle comme le saut de ligne ou la marque d'un nouveau paragraphe (U+2029 en Unicode, mais ce séparateur de paragraphe n'est que rarement géré par les logiciels).

À l'autre extrémité se trouve les formats binaires : lisibles et modifiables uniquement par des applications spécifiques (et c'est pour cela qu'ils sont distribués avec le type MIME `application/quelquechose`).

Entre les deux se trouvent le texte formaté ou marqué. Il s'agit à première vue de texte brut mais il inclut en fait quelques marques qui ont une signification non prévue par le jeu de caractères. Un exemple classique est la mise en évidence qui, en Markdown, se fait en encadrant le texte avec des étoiles : « il est

1. Pour voir le RFC de numéro NNN, `https://www.ietf.org/rfc/rfcNNN.txt`, par exemple `https://www.ietf.org/rfc/rfc7763.txt`

très important de comprendre que... » Le texte reste lisible sans logiciel spécifique (d'autant plus qu'ici, Markdown reprend une ancienne convention d'écriture largement utilisée sur l'Internet). Et n'importe quel éditeur (et n'importe quel clavier) permet de créer un tel texte.

On sépare souvent les formats de texte marqués en formats stricts ("*markup*") et formats légers ("*lightweight markup*"). Les premiers sont plus proches des formats binaires, dans la mesure où ils ont des marques bien séparées du texte, et suivent une syntaxe rigide. Ce sont par exemple XML et LaTeX. Cette syntaxe rigide permet de vérifier la qualité technique du document et mène à une meilleure interopérabilité. Mais elle peut être pénible pour les utilisateurs débutants ou occasionnels. Plus proches du texte brut, les formats de texte marqué légers ont des marques plus discrètes, et ont souvent une syntaxe moins formelle, plus tolérante (un cas extrême est Markdown, qui ne connait même pas la notion d'erreur de syntaxe). Ils sont conçus pour être facilement accessibles au nouvel utilisateur (ou à celui qui n'a pas pratiqué depuis longtemps). Jeff Atwood qualifiait ces formats légers d'« humains <http://blog.codinghorror.com/is-html-a-humane-markup-language/> ».

Markdown est issu d'un langage originellement développé par John Gruber (et Aaron Swartz). Il a été rapidement un grand succès, notamment pour les tâches simples (comme de saisir un commentaire ou un rapport de bogue dans un formulaire Web). Et il a connu de nombreuses extensions. Beaucoup de programmes et de langages savent traiter le Markdown, chacun avec sa **variante** particulière.

Markdown est clairement dans le camp de l'« **informalité** ». Un des événements fondateurs fut un problème avec de nombreux flux de syndication rendus invalides <http://daringfireball.net/2004/03/dive_into_markdown> juste par un article contenant du code incorrect (la norme XML <http://www.w3.org/TR/2008/REC-xml-20081126#dt-fatal> dit bien qu'une fois qu'une erreur apparait dans un fichier XML, le processeur ne doit pas continuer). Avec Markdown, cela ne risque pas d'arriver, tout texte est du Markdown légal. Cela ressemble au fameux principe de robustesse <https://www.bortzmeyer.org/principe-robustesse.html> des protocoles réseau. (À noter que ce principe a été suivi, en pratique, par la plupart des navigateurs Web, lorsqu'il s'agit d'analyser du HTML. Résultat, les auteurs se sont habitués à taper n'importe quoi et le Web est aujourd'hui une poubelle de fichiers HTML bourrés d'erreurs.) Lorsqu'on soumet un fichier Markdown à un processeur, cela « marche » toujours. Si on n'est pas satisfait du résultat, on modifie le source et on réessaie (cf. « *"The Talk Show : Ep. 88 : 'Cat Pictures' (Side 1)"* <http://daringfireball.net/thetalkshow/2014/07/19/ep-088> ».) Pas de processus de validation, comme en XML ou HTML strict, seul le résultat compte.

Malgré ou bien à cause de ce principe, Markdown est un grand succès. On le trouve utilisé partout et il est probablement le plus populaire des langages de marquage « légers ». On peut l'éditer avec n'importe quel logiciel et lire le source, même sans avoir subi aucun apprentissage de sa syntaxe. Des gens utilisent Markdown pour des articles scientifiques <http://blogs.law.harvard.edu/pamphlet/2014/08/29/switching-to-markdown-for-scholarly-article-production/>, l'écriture de scénarios <http://fountain.io/>, ou pour faire des maths <https://github.com/cben/mathdown/wiki/math-in-markdown>, le domaine qui était autrefois le bastion de LaTeX. (Personnellement, je m'en sers pour les rapports de bogue sur GitHub, les README de logiciels, les rapports techniques internes à mon employeur, LaTeX ou DocBook prenant le relais pour les choses plus « sérieuses ». Markdown avec les sites qui le gèrent permet aussi de faire du mini-blogging très simplement et très rapidement <https://gist.github.com/bortzmeyer/1493af2e586542e4861c>. Regardez aussi le témoignage de Yann Houry <https://medium.com/@yannhoury/utiliser-le-markdown-881f8800aff0>.

Depuis le RFC 7763, il existe un type MIME pour identifier Markdown, `text/markdown`. (Une extension de nom de fichier comme `.md` ne suffit pas : tout le contenu n'est pas forcément dans des fichiers.) Un seul type MIME décrit toutes les variantes puisque n'importe quel processeur pourra traiter n'importe quelle variante (même si le résultat ne sera pas toujours celui voulu.) À noter que la plupart

des variantes de Markdown n'ont pas de mécanisme pour mettre les métadonnées : il faut les placer à l'extérieur.

Comment préserver ce type MIME lorsque du contenu Markdown est copié sur l'Internet ? Parfois, la copie ne préserve pas cette information et Markdown n'a pas l'équivalent du `<?xml version="1.0">` qui, au début des contenus en XML, indique sans ambiguité qu'il s'agit de XML (il a été proposé de l'introduire mais cela peut casser les mécanismes de métadonnées existants). Utiliser le contexte (le répertoire où se trouve un fichier, par exemple...) est casse-gueule car, justement, l'envoi d'un document fait facilement perdre le contexte. La section 2 du RFC propose et discute plusieurs stratégies. Utiliser le nom de fichier, lorsqu'on passe par des fichiers, est possible. `mon-roman.md` est un texte en Markdown, on le sait par l'extension. Et la variante ? On peut l'indiquer avant, par exemple `mon-roman.pandoc.md` si le texte est écrit dans la variante Pandoc. Si le système de fichiers permet de stocker des attributs ou métadonnées, c'est une autre bonne solution.

Mais il y en a d'autres. Une solution, suggérée par le RFC 6533, est de stocker les métadonnées et notamment l'en-tête MIME dans un fichier séparé, conventionnellement nommé. Si le fichier texte est `mon-roman.md`, le fichier des métadonnées pourrait être `mon-roman.md.headers`.

Ou alors dans le même fichier ? L'idée est de transformer le fichier Markdown en un fichier au format IMF ("*Internet Message Format*", spécifié dans le RFC 5322) en mettant des en-têtes d'abord, puis le contenu en Markdown :

```
MIME-Version: 1.0
Content-Type: text/markdown; charset=utf-8
Date: Wed, 10 Feb 2016 20:59:06 +0100 (CET)

Exemple d'un fichier « armé »
=============================

Notez la ligne vide qui sépare les en-têtes du corps.

Il est *important* que son extension ne soit pas .md ou .markdown, ce
n'est plus du Markdown pur mais de l'IMF, son extension doit donc être
.eml ou .msg.

...
```

Le RFC contient plein d'autres stratégies rigolotes pour résoudre ce problème. Par exemple, on pourrait, lors de la réception, voire de l'envoi, d'un contenu en Markdown, générer/envoyer un script qui contienne les commandes permettant de le traiter convenablement. Ou on pourrait carrément envoyer le résultat du traitement, et pas le Markdown : si on veut juste que le destinataire puisse lire, pas la peine de lui envoyer le « source », on pourrait juste lui transmettre le XHTML (je ne suis pas sûr d'être d'accord avec cette curieuse idée : un des gros avantages de Markdown est qu'il est lisible même si on n'a aucun logiciel spécifique, et si on n'a jamais entendu parler de ce format).

Cette section 2 se termine avec le cas des VCS. Subversion offre un excellent mécanisme de stockage et de versionnement des méta-données. Il suffit donc de définir la métadonnée standard `svn:mime-type` et on ne risque plus de perdre cette information :

```
% svn propset svn:mime-type text/markdown mon-roman.md
property 'svn:mime-type' set on 'mon-roman.md'

% svn commit -m "Type indiqué" mon-roman.md
...
```

Hélas, git n'a, lui, aucun moyen propre de stocker les métadonnées. Il faut donc recourir à une autre des stratégies mentionnées plus haut (fichier d'en-têtes avec un nom conventionnel, extension du fichier, etc).

La section 3 de notre RFC est consacrée aux formulaires d'enregistrement des différentes variantes de Markdown, désormais mémorisées à l'IANA <https://www.iana.org/assignments/markdown-variants/markdown-variants.xml>. Cette section est utile si vous voulez vous renseigner sur une variante particulière, mais aussi si vous avez développé votre propre variante (sans doute une mauvaise idée mais les programmeurs adorent réinventer la roue) et que vous voulez l'enregistrer à l'IANA. Vous pouvez alors utiliser les cas de cette section comme exemples concrets, avant d'écrire votre propre demande.

Prenons par exemple la variante que j'utilise le plus souvent, `Pandoc`. Contrairement au Markdown originel, elle ne vise pas uniquement la génération de HTML mais également des tas d'autres formats de sortie. Elle a des tas d'extensions par rapport à l'original, comme les tableaux ou bien les notes de bas de page. Sa fiche contient :

```
Identifier: pandoc

Name: Pandoc

...

Extensions to turn off (on by default):

                escaped_line_breaks
                blank_before_header
                header_attributes
...

References:
<http://johnmacfarlane.net/pandoc/README.html#pandocs-markdown>

Contact Information:
  (individual) Prof. John MacFarlane <jgm@berkeley.edu>
                           <http://johnmacfarlane.net/>
```

Les autres variantes enregistrées via cette section 3 sont MultiMarkdown, GitHub <https://help.github.com/articles/working-with-advanced-formatting/>, Fountain, CommonMark <http://commonmark.org/>, "*Markdown for RFCs*", "*Pandoc for RFCs*" (RFC 7328) et PHP Markdown (utilisé par plusieurs CMS comme Drupal).

La section 4 de notre RFC fournit des exemples de ces différentes variantes et de ce qu'elles apportent. Par exemple, la variante GitHub apporte la transformation automatique des URL en liens hypertexte.