RFC 6266 : Use of the Content-Disposition Header Field in the Hypertext Transfer Protocol (HTTP)

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 6 juin 2011

Date de publication du RFC : Juin 2011

https://www.bortzmeyer.org/6266.html

Le protocole HTTP ne sert pas qu'à récupérer des pages HTML qu'on va afficher à l'utilisateur. Une part importante de ses usages est la récupération de fichiers qu'on enregistrera sur le stockage local. Quel nom donner au fichier local ? HTTP dispose depuis longtemps d'un en-tête dans les réponses, `Content-Disposition:`, qui fournissait des suggestions de noms. Cet en-tête était imparfaitement normalisé et traité un peu à part. Ce nouveau RFC en fait un élément standard et rigoureusement défini de la panoplie HTTP.

Cet en-tête était donc décrit dans les RFC 2616 [1] (section 19.5.1, qui prend des pincettes et affirme que cet en-tête n'est décrit que parce qu'il est largement mis en œuvre mais qu'il ne fait pas vraiment partie de la norme) et RFC 2183. La description était incomplète, notamment question internationalisation.

La grammaire formelle de cet en-tête est désormais décrite en section 4. Voici un exemple :

```
Content-Disposition: Attachment; filename=example.txt
```

1. Pour voir le RFC de numéro NNN, `https://www.ietf.org/rfc/rfcNNN.txt`, par exemple `https://www.ietf.org/rfc/rfc2616.txt`

Cette réponse HTTP demande que le fichier soit traité comme un attachement, à enregistrer, et que le nom suggéré est `example.txt`. Le premier terme se nomme le **type** (section 4.2) et à la place du `Attachment` qu'on a ici, on pourrait avoir `Inline` qui suggère au client HTTP d'afficher le fichier, plutôt que de l'enregistrer (c'est le comportement par défaut en l'absence de l'en-tête `Content-Disposition`). Les types et les noms de paramètres (ci-dessous) sont insensibles à la casse.

Après le type et un point-virgule, on trouve des **paramètres**. Ici, le paramètre `filename` était indiqué, pour proposer un nom de fichier. À noter que ce paramètre peut être complété par un `filename*`, doté de possibilités étendues, notamment celle de suggérer des noms comportant des caractères non-Latin1 (cf. RFC 8187, section 6 et annexe C). Ce `filename*` est plus récent et pas compris de tous les clients HTTP. L'idée est que le serveur moderne envoie les deux, le client ancien ne gardera que `filename` (qui est « sûr ») et le client nouveau n'utilisera que `filename*`. Voici un exemple suivant ces recommandations :

```
Content-Disposition: attachment;
                     filename="EURO rates";
                     filename*=utf-8''%e2%82%ac%20rates
```

Ce fichier sera enregistré, par un client récent, sous le nom € `rates`.

J'ai utilisé à plusieurs reprises des verbes comme « suggérer » ou « proposer » car le client HTTP reste maître du choix du nom. Il est même important qu'il n'accepte pas aveuglément n'importe quel nom donné par le serveur (imaginez une machine Unix où le navigateur Web stockerait le fichier sous le nom `/etc/passwd` sans réfléchir). Plus précisement, insiste le RFC (sections 4.3 et 7), le client HTTP doit :

— N'autoriser l'écriture du fichier que dans les dossiers ou répertoires choisis. Le mieux est d'ignorer complètement tout chemin inclus dans le nom (ne garder que le nom de fichier proprement dit).
— Se méfier de l'extension (dans le premier exemple, `.txt`). Avec certains systèmes d'exploitation comme Windows, c'est cette extension (et pas le type MIME - RFC 2046 - envoyé avec le fichier) qui détermine le comportement associé au fichier. Un nom se terminant en `.exe` peut donc avoir des conséquences ennuyeuses.
— Certains caractères sont spéciaux pour le système d'exploitation qui va stocker le fichier (comme le point-virgule ou le tube sur Unix) et il vaut donc mieux les supprimer avant d'enregistrer. Même chose, bien que moins dangereux, pour des noms qui pourraient perturber l'interface utilisateur, comme les espaces pour Unix : ils sont légaux mais pas pratiques à gérer depuis le shell.

Un bon exemple de faille de sécurité provoquée par le non-respect de ces règles figure en <enhttp://securitytracker.com/id?1024583>.

La section 4.4 ajoute qu'il faut ignorer les paramètres inconnus (pour permettre l'extensibilité ultérieure). Les en-têtes invalides devraient être ignorés par les clients HTTP. De nombreux détails sur le traitement de `Content-Disposition:` par les clients figurent dans l'excellente page <http://greenbytes.de/tech/tc2231/>.

Les clients HTTP, par exemple les navigateurs Web, respectent-ils cette norme ? Actuellement, Firefox, Opera et Konqueror le font, ainsi que Chrome depuis sa version 9 (parait-il : ça ne marche pas chez moi) et Internet Explorer depuis sa version 9. Cela ne signifie pas qu'ils gèrent tout et notamment Firefox 3 semble ignorer le paramètre de nom de fichier international, `filename*` (bogue #588781 <https://bugzilla.mozilla.org/show_bug.cgi?id=588781>). wget semble ignorer l'indication de nom de fichier par défaut. Il faut lui ajouter l'option `--content-disposition` pour que cela marche. C'est encore pire pour curl dont l'auteur refuse <http://sourceforge.net/tracker/?func=detail&atid=100976&aid=826813&group_id=976> de gérer `Content-Disposition:`

et fait semblant de croire que l'option `--remote-name` le remplace. lynx a une amusante bogue : il inclus le point-virgule qui suit (s'il y a un deuxième paramètre), dans le nom de fichier.

À noter que le registre des en-têtes <https://www.iana.org/assignments/message-headers/perm-headers.html> (section 8.2) et celui des paramètres <https://www.iana.org/assignments/mail-cont-disp/mail-cont-disp.xml> de `Content-Disposition:` (section 8.1) sont utilisés par d'autres protocoles que HTTP (cf. section 4.5).

L'annexe A résume les changements par rapport au RFC 2616, le premier qui mentionnait cet en-tête `Content-Disposition:` dans le cadre de HTTP. Le principal est l'ajout des capacités d'internationalisation du RFC 8187. Mais il y a aussi quelques détails, comme la suppression de la restriction qui limitait cet en-tête aux ressources de type `application/octet-stream`. L'annexe B, elle, résume les changements depuis le RFC 2183. Ils consistent essentiellement en l'abandon de paramètres intéressants mais qui, en pratique, n'ont jamais été mis en œuvre par les clients HTTP, comme `creation-date` ou `size`.

L'annexe C discute l'approche d'internationalisation utilisée dans ce RFC, en comparant avec les alternatives. Un peu d'histoire : les en-têtes HTTP sont historiquement limités à ISO 8859-1 (RFC 2616, section 2.2 ; j'ai bien dit ISO 8859-1 et pas ASCII ; c'est ainsi que ce blog émettait un en-tête en ISO 8859-1, qui avait planté quelques logiciels qui n'avaient pas bien lu le RFC 2616 ; le RFC 7230, depuis, a supprimé cette possibilité). C'est évidemment une limite unsupportable. La solution standard, depuis belle lurette (RFC 2231, en 1997) est d'encoder les caractères Unicode en « notation pourcent ». Le RFC 8187, plusieurs fois cité ici, est l'application de ce principe à HTTP. Mais les clients HTTP, notamment les navigateurs, ont très souvent essayé des approches non-standard. Pourquoi n'ont-elles pas été reconnues dans notre RFC 6266 ?

Par exemple, certains ont implémenté les encodages du RFC 2047, même si ce RFC précisait bien qu'ils ne devaient pas être utilisés dans les en-têtes. D'autres gèrent l'encodage pourcent (RFC 3986, section 2.1) dans le paramètre `filename`, ce qui encourage les serveurs à l'utiliser ainsi, semant la confusion chez les navigateurs. Ainsi, si le nom de fichier est `caf%C3%A9.html`, certaines navigateurs l'enregistreront sous le nom `café.html` et d'autres sont le nom `caf%C3%A9.html`. Plus rigolo, certains navigateurs pratiquent l'examen du nom du fichier et appliquent diverses heuristiques pour déterminer son contenu (« Hmmm, on dirait de l'UTF-8 encodé pour-cent... Je vais essayer ça. ») C'est évidemment très fragile. Le tableau « "*Test Result Summary*" » en <http://greenbytes.de/tech/tc2231/> résume fort bien le comportement actuel des navigateurs Web.

Les programmeurs, enfin, ont tout intérêt à lire l'annexe D, qui rassemble les conseils pour les développeurs de serveurs HTTP et d'applications Web (en pratique, c'est souvent le "*framework*" de développement, pas le serveur HTTP, qui génère ces en-têtes). Parmi les conseils :

— Inclure un paramètre `filename` si l'ASCII suffit (ce devrait être Latin-1 mais le RFC note qu'il n'est pas prudent de compter dessus, trop de programmeurs n'ont pas lu le RFC 2616 et ne gèrent pas correctement le non-ASCII et, de toute façon, le RFC 7230 a modifié la règle depuis),
— Inclure un paramètre `filename*` si l'ASCII ne suffit pas, tout en sachant que bien des clients Web ne le cherchent pas et n'utiliserons que `filename` ; le RFC suffère donc de mettre les deux, une version dégradée du nom étant présente dans `filename` (cela est possible en français, voir l'exemple ci-dessous, mais pas tellement en arabe),
— Utiliser uniquement UTF-8 comme encodage de `filename*`.

Notez que le moteur de recherche de mon blog <https://www.bortzmeyer.org/moteur-recherche.html> produit des en-têtes « `Content-Disposition:` » en respectant, je l'espère, les conseils de cette annexe D...

```
% wget --server-response 'http://www.bortzmeyer.org/search?pattern=café chocolat&format=atom'
...
  Server: Apache/2.2.9 (Debian) mod_wsgi/2.5 Python/2.5.2
  Content-Disposition: Inline; filename=bortzmeyer-search-caf--chocolat.atom; filename*=UTF-8''bortzmeyer-se
  Content-Type: application/atom+xml; charset="utf-8"
...
```

(Si vous lisez le code source <https://www.bortzmeyer.org/blog-implementation.html>, c'est dans `wsgis/search.py`.)