RFC 5051 : i;unicode-casemap - Simple Unicode Collation Algorithm

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 24 octobre 2007. Dernière mise à jour le 30 octobre 2007

Date de publication du RFC : Octobre 2007

https://www.bortzmeyer.org/5051.html

Plusieurs protocoles, notamment IMAP, ont besoin de **trier** des chaînes de caractères. Il existe plusieurs règles de tri, plus ou moins faciles à mettre en œuvre et plus ou moins « correctes » pour l'utilisateur et ce RFC spécifie une règle qui respecte mieux les chaînes Unicode, sans pour autant être parfaite du point de vue des règles des langues vivantes.

Pour éviter à chaque application de réinventer la roue, il existe un registre IANA <https://www.iana.org/assignments/collation/collation-index.html> des règles de tri, que les protocoles n'ont qu'à référencer. Ce registre, initialement créé par le RFC 4790[1], se remplit petit à petit. Les premières règles étaient triviales à implémenter (comme `i;octet`, qui travaille uniquement au niveau binaire, sans comprendre les caractères), mais très éloignées de ce qu'un utilisateur humain appelerait un tri. Notre RFC normalise une nouvelle règle, `i;unicode-casemap` qui fournit un tri indépendant de la casse pour les chaînes Unicode.

La règle est assez simple à exprimer (et le RFC est très court) :

— Convertir les caractères dans leur version capitalisée (attention, contrairement à ASCII, on ne peut pas le faire algorithmiquement, il faut une table),
— Décomposer chaque caractère (pratiquement la normalisation Unicode NFKD).

1. Pour voir le RFC de numéro NNN, https://www.ietf.org/rfc/rfcNNN.txt, par exemple https://www.ietf.org/rfc/rfc4790.txt

Notre RFC donne l'exemple du caractère « "*LATIN CAPITAL LETTER DZ WITH CARON*" » (U+01C4 en Unicode, soit [Caractère Unicode non montré [2]]. Il est capitalisé en « "*LATIN CAPITAL LETTER D WITH SMALL LETTER Z WITH CARON*" (U+01C5, soit [Caractère Unicode non montré]). Il est ensuite décomposé en « "*LATIN CAPITAL LETTER D*" » (un grand D ordinaire) et « "*LATIN SMALL LETTER Z WITH CARON*" » (U+017E soit [Caractère Unicode non montré]). Ce dernier est à son tour décomposé et le résultat final est Dz[Caractère Unicode non montré] (trois caractères Unicode, le dernier étant le caron combinant, U+030C, dites-moi si vous voyez tout correctement sur votre navigateur). On peut alors trier.

Utilisant la base de données Unicode <`https://www.bortzmeyer.org/unicode-to-sql.html`>, on peut programmer cet algorithme (ici un programme (en ligne sur `https://www.bortzmeyer.org/files/unicode-decompose.py`) écrit en Python) facilement. Voici les résultats, avec successivement le « e accent aigü », le « grand dz avec caron », et la ligature arabe « Djalladjalalouhou » :

```
 % ./decompose.py --rfc5051 -n U+00E9
U+000045 U+000301
% ./decompose.py --rfc5051 -n U+1C4
U+000044 U+00007A U+00030C
% ./decompose.py --rfc5051 -n U+FDFB
U+00062C U+000644 U+000020 U+00062C U+000644 U+000627 U+000644 U+000647
```

Le programme n'est pas 100 % parfait (il ne gère pas le cas où la capitalisation d'une lettre est une chaîne et pas une lettre unique) mais presque.

Notons enfin une très intéressante section 3 sur la sécurité, qui explique comment deux implémentations correctes de cet algorithme peuvent quand même donner des résultats différents (un problème fréquent avec Unicode).

2. Car trop difficile à faire afficher par LaTeX