RFC 3492 : Punycode: A Bootstring encoding of Unicode for Internationalized Domain Names in Applications (IDNA)

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 19 octobre 2007

Date de publication du RFC : Mars 2003

http://www.bortzmeyer.org/3492.html

La norme IDNA, qui permet d'utiliser des caractères Unicode dans les noms de domaine, dépend de deux algorithmes spécifiés séparément, nameprep (RFC 3491 [1]) et Punycode (notre RFC), qui transforme une chaîne Unicode en un sous-ensemble d'ASCII, qui peut être utilisé dans des logiciels non-Unicode.

En effet, les applications qui gèrent des noms de machine sont tenues aux règles très restrictives du RFC 1123, qui limite ces noms à **LDH**, un sous-ensemble d'ASCII limité aux lettres, chiffres et au tiret. (Contrairement à ce qu'on lit souvent, cette restriction n'a rien à voir avec le DNS, qui accepte bien plus de caractères, depuis toujours.) La solution adoptée dans le cadre IDNA est donc d'encoder les caractères Unicode en Punycode, qui n'utilise que LDH. Punycode est donc un concurrent d'autres encodages d'Unicode, plus généralistes, comme UTF-8 ou UTF-7.

Punycode est spécifié en deux temps. Un algorithme général, Bootstring, présenté dans les sections 3 et 4, permet de mettre en correspondance une chaîne de caractères Unicode avec une chaîne de caractères écrite dans un sous-ensemble d'Unicode. Cet algorithme est **paramétré** par certaines variables, qui reçoivent ensuite en section 5 une valeur particulière, ces valeurs définissant le profil Punycode. Notons qu'à ma connaissance, il n'existe aucun autre profil de Bootstring. Donc, en pratique, il n'est pas trop gênant de confondre Bootstring et Punycode.

La section 1.1 de notre RFC détaille les propriétés de Bootstring. Il est **complet** (toute chaîne Unicode peut être représentée en Punycode mais notons qu'IDNA, la seule application de Bootstring aujourd'hui, interdit certains caractères comme les espaces). Et il est **sans perte**, toute chaîne Unicode encodée en Punycode peut être récupérée par un décodage (là encore, IDNA a un mécanisme de normalisation,

1. Pour voir le RFC de numéro NNN, https://www.ietf.org/rfc/rfcNNN.txt, par exemple https://www.ietf.org/rfc/rfc3491.txt

nameprep - RFC 3491 - qui n'a pas cette propriété). Punycode est également **compact**, un gros avantage vue la limitation des **labels** DNS à 63 caractères.

Punycode n'est pas un algorithme très compliqué, le RFC est assez court (et une bonne partie est occupée par l'implémentation de référence). Mais il contient davantage d'algorithmique que beaucoup de RFC.

On l'a dit, la section 5 fixe les paramètres spécifiques à Punycode. Par exemple, le paramètre `base` est égal à 36 à cause de LDH (vingt-six lettres et dix chiffres, le tiret ayant un autre usage).

Passons maintenant à un exemple concret. Avec le code C inclus dans le RFC (un bel exemple de C ANSI, qui ne soulève aucune objection du sourcilleux compilateur GNU, même avec les options `-Wall` et `-pedantic`), convertissons le mot « café » en Punycode (ce programme ne lit les caractères Unicode que sous la forme U+nnnn) :

```
% gcc -o punycode punycode.c
% echo u+0063 u+0061 u+0066 u+00E9 | ./punycode -e
caf-dma
```

Le mot d'origine étant essentiellement composé d'ASCII, on en reconnait une partie dans la version Punycode. Traduisons-le maintenant en sens inverse :

```
% echo caf-dma | ./punycode -d
u+0063
u+0061
u+0066
u+00E9
```

On voit qu'on a récupéré, comme prévu, exactement la même chaîne.