RFC 9285 : The Base45 Data Encoding

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 10 août 2022

Date de publication du RFC : Août 2022

https://www.bortzmeyer.org/9285.html

Ce RFC décrit un encodage en texte de données binaires, l'encodage Base45. Proche conceptuellement d'encodages comme Base64, il est, par exemple, beaucoup utilisé pour le code QR.

Ces codes QR ne permettent pas de stocker directement des données binaires quelconques car le contenu sera toujours interprété comme du texte. Il est donc nécessaire d'encoder et c'est le rôle de Base45. Il existe bien sûr d'innombrables autres mécanismes d'encodage en texte, comme Base16, Base32 et Base64, spécifiés dans le RFC 4648 [1] mais Base45 est plus efficace pour les codes QR, qui réencodent ensuite.

Base45 utilise un sous-ensemble d'ASCII, de 45 caractères (d'où son nom). Deux octets du contenu binaire sont encodés en trois caractères de ce sous-ensemble. Par exemple, deux octets nuls donneront la chaine de caractères "000". Regardons tout de suite avec le module Python base45 <https://pypi.org/project/base45/> :

```
% pip3 install base45
% python3
>>> import base45
>>> base45.b45encode(b'Bonjour, tout le monde')
b'.H86/D34ENJES4434ESUETVDL44-3E6VC'
>>>
```

Et décodons avec le module Elixir base45 <https://framagit.org/bortzmeyer/base45> :

1. Pour voir le RFC de numéro NNN, https://www.ietf.org/rfc/rfcNNN.txt, par exemple https://www.ietf.org/rfc/rfc4648.txt

```
% iex -S mix
iex(1)> Base45.decode(".H86/D34ENJES4434ESUETVDL44-3E6VC")
"Bonjour, tout le monde"
```

C'est parfait, tout le monde est d'accord. Ici, évidemment, la chaine originale n'avait pas vraiment besoin d'être encodée donc on va essayer avec du binaire, les trois octets 42, 1 et 6, encodage en Elixir, décodage en Python :

```
iex(1)> Base45.encode(<<42, 1, 6>>)
"/D560"

>>> base45.b45decode("/D560")
b'*\x01\x06'
```

Encore une fois, tout va bien, Elixir a encodé les trois octets du binaire en quatre caractères, que Python a su décoder (l'astérisque est affiché car son code ASCII est 42).

Je vous laisse découvrir l'algorithme complet (il est assez simple) dans la section 3 du RFC. Si vous le programmez vous-même (ce qui n'est sans doute pas une bonne idée, il existe déjà de nombreuses mises en œuvre), attention aux cas limites comme un binaire d'un seul octet, ou comme une chaine à décoder qui compte des caractères invalides. Ici, un essai avec un caractère invalide, le signe égal :

```
% python3
>>> import base45
>>> base45.b45decode("AAAA=")
Traceback (most recent call last):
  File "/home/stephane/.local/lib/python3.8/site-packages/base45/__init__.py", line 30, in b45decode
    buf = [BASE45_DICT[c] for c in s.rstrip("\n")]
  File "/home/stephane/.local/lib/python3.8/site-packages/base45/__init__.py", line 30, in <listcomp>
    buf = [BASE45_DICT[c] for c in s.rstrip("\n")]
KeyError: '='

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/home/stephane/.local/lib/python3.8/site-packages/base45/__init__.py", line 54, in b45decode
    raise ValueError("Invalid base45 string")
ValueError: Invalid base45 string
```

Pensez à tester votre code avec de nombreux cas variés. Le RFC cite l'exemple des chaines "FGW" (qui se décode en une paire d'octets valant chacun 255) et "GGW" qui, quoique proche de la précédente et ne comportant que des caractères de l'encodage Base45 est néanmoins invalide (testez-la avec votre décodeur). Le code de test dans le module Elixir `<https://framagit.org/bortzmeyer/base45>` donne des idées de tests utiles (certains viennent de bogues détectées pendant le développement). Regardez `test/base45_test.exs`.

Et, comme toujours, lorsque vous recevez des données venues de l'extérieur, soyez paranoïaques dans le décodage ! Un code QR peut être malveillant et chercher à activer une bogue dans le décodeur (section 6 du RFC).

Enfin, le RFC rappelle qu'une chaine encodée n'est pas forcément directement utilisable comme URL, elle peut comporter des caractères qui ne sont pas sûrs, il faut donc encore une étape d'encodage si on veut en faire un URL.

Compte tenu de l'importance des codes QR, on trouve des mises en œuvre de Base45 un peu partout. J'ai par exemple cité celle en Python <https://github.com/kirei/python-base45>. Si vous programmez en Elixir, il y a une bibliothèque sur Hex <https://hex.pm/packages/base45> (écrite en Erlang) mais aussi ma bibliothèque <https://hex.pm/packages/base45_elixir> (aux performances très basses).

Sinon, si vous cherchez un bon article d'explication sur Base45, je recommande celui-ci <https://billatnapier.medium.com/so-what-is-base-45-and-where-is-it-used-1ab53279d705>.