

# RFC 4934 : Extensible Provisioning Protocol (EPP) Transport Over TCP

Stéphane Bortzmeyer  
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 1 Avril 2008. Dernière mise à jour le 7 Mai 2008

Date de publication du RFC : Mai 2007

<http://www.bortzmeyer.org/4934.html>

---

Ce court RFC spécifiait comment utiliser le protocole d'avitaillement EPP au dessus d'une simple connexion TCP. Il a été remplacé depuis par le RFC 5734<sup>1</sup>.

EPP, décrit dans le RFC 4930 est à sa base uniquement un format XML pour les requêtes d'avitaillement (création, mise à jour et destruction d'objets) et leurs réponses. Ces éléments XML peuvent être transmis de différence façon (au dessus de HTTP, de BEEP, par courrier électronique, etc), et notre RFC normalise la plus commune aujourd'hui, une simple connexion TCP. Il remplace le RFC 3734, avec uniquement des modifications de détail, portant notamment sur la sécurité (section 8).

Le RFC est court car il n'y a pas grand'chose à dire, juste l'utilisation des primitives de TCP (ouverture et fermeture de connexion, section 2 du RFC), l'ordre des messages (section 3) et le fait que chaque élément EPP soit précédé d'un entier qui indique sa taille (section 4). Sans cet entier (qui joue le même rôle que l'en-tête `Content-Length` de HTTP), il faudrait, avec la plupart des implémentations, lire les données octet par octet (sans compter que la plupart des analyseurs XML ne savent pas analyser de manière incrémentale, il leur faut tout l'élément). En outre, sa présence permet de s'assurer que toutes les données ont été reçues (voir l'excellent article "*The ultimate SO\_LINGER page, or : why is my tcp not reliable*" ([http://blog.netherlabs.nl/articles/2009/01/18/the-ultimate-so\\_linger-page-or-why-is-my-tcp-not-reliable](http://blog.netherlabs.nl/articles/2009/01/18/the-ultimate-so_linger-page-or-why-is-my-tcp-not-reliable))).

L'entier en question est fixé à 32 bits. Si on programme un client EPP en Python, l'utilisation brutale du module `struct` (<http://docs.python.org/lib/module-struct.html>) ne suffit pas forcément. En effet :

---

<sup>1</sup>Pour voir le RFC de numéro NNN, <http://www.ietf.org/rfc/rfcNNN.txt>, par exemple <http://www.ietf.org/rfc/rfc5734.txt>

```
struct.pack("I", length)
```

force un entier (`int`) mais pas forcément un entier de 32 bits. Pour forcer la taille, il faut utiliser également, comme précisé dans la documentation, les opérateurs `j` et `i`, qui servent aussi à forcer la boutianité (merci à Kim-Minh Kaplan pour son aide sur ce point). Voici une démonstration (un "I" standard fait 4 octets alors que le type long de C peut faire 4 ou 8 octets) :

```
# Machine 32 bits :
```

```
Python 2.4.4 (#2, Apr 5 2007, 20:11:18)
[GCC 4.1.2 20061115 (prerelease) (Debian 4.1.1-21)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import struct
>>> print struct.calcsize("l")
4
>>> print struct.calcsize(">l")
4
```

```
# Machine 64 bits :
```

```
Python 2.4.5 (#2, Mar 11 2008, 23:38:15)
[GCC 4.2.3 (Debian 4.2.3-2)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import struct
>>> print struct.calcsize("l")
8
>>> print struct.calcsize(">l")
4
```

Si on a quand même un doute, on peut tester la taille obtenue mais ce code est probablement inutile (merci à David Douard pour son aide ici) :

```
# Get the size of C integers. We need 32 bits unsigned.
format_32 = ">I"
if struct.calcsize(format_32) < 4:
    format_32 = ">L"
    if struct.calcsize(format_32) != 4:
        raise Exception("Cannot find a 32 bits integer")
elif struct.calcsize(format_32) > 4:
    format_32 = ">H"
    if struct.calcsize(format_32) != 4:
        raise Exception("Cannot find a 32 bits integer")
else:
    pass
...
def int_from_net(data):
    return struct.unpack(format_32, data)[0]

def int_to_net(value):
    return struct.pack(format_32, value)
```

L'algorithme complet d'envoi est :

---

<http://www.bortzmeyer.org/4934.html>

```
epp_as_string = ElementTree.tostring(epp, encoding="UTF-8")
# +4 for the length field itself (section 4 mandates that)
# +2 for the CRLF at the end
length = int_to_net(len(epp_as_string) + 4 + 2)
self._socket.send(length)
self._socket.send(epp_as_string + "\r\n")
```

et la lecture :

```
data = self._socket.recv(4) # RFC 4934, section 4, the length
                             # field is 4 bytes long
length = int_from_net(data)
data = self._socket.recv(length-4)
epp = ElementTree.fromstring(data)
if epp.tag != "{%s}epp" % EPP.NS:
    raise EPP_Exception("Not an EPP instance: %s" % epp.tag)
xml = epp[0]
```

Le code Python complet (qui ne met en œuvre qu'une petite partie de EPP, le but était juste de tester ce RFC 4934), utilisant la bibliothèque ElementTree (<http://effbot.org/zone/element-index.htm>), est disponible en ligne (en ligne sur <http://www.bortzmeyer.org/files/epp-python.tar.gz>).