RFC 7464 : JavaScript Object Notation (JSON) Text Sequences

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 26 février 2015

Date de publication du RFC : Février 2015

https://www.bortzmeyer.org/7464.html

Un document JSON est normalement formé d'un texte JSON tel qu'un tableau ou un objet. Mais certains usages de JSON auraient besoin d'un autre genre de documents, une suite de textes JSON, séparés par un caractère de séparation bien défini. C'est ce que normalise ce RFC. Le nouveau type de document, `application/json-seq`, est formé d'une suite de textes JSON, séparés par les caractères ASCII U+001E, alias RS ("*record separator*") et U+000A, alias LF ("*line feed*").

JSON est normalisé dans le RFC 8259[1]. Il est très utilisé dans un grand nombre de contextes. Du fait qu'un document JSON est un texte JSON (par exemple un objet ou un tableau), une grande partie des analyseurs JSON lisent tout et mettent tout en mémoire avant de le passer à l'application. Ce n'est pas bien adapté au cas où on écrit du JSON dans un journal, par exemple, et où on ne connait pas à l'avance la fin du document. Souvent, il n'y a pas à proprement parler de fin, et on ne peut donc pas utiliser un tableau car on ne saurait pas où mettre le] final. Pire, imaginons une séquence d'un million d'entrées, chacune faisant un kilo-octet. Avec un analyseur ordinaire, cela ferait un giga-octet à mettre en mémoire, une quantité non triviale. Certes, il existe des analyseurs en flux ("*streaming*"), mais ils ne sont pas très répandus, et notre RFC les trouve difficiles d'utilisation.

D'où l'idée de ce RFC : une séquence de textes JSON, qui n'est pas elle-même un texte JSON, et qui peut être produite et consommée de manière incrémentale.

La définition rigoureuse figure en section 2. À noter que deux grammaires ABNF sont données, une pour les générateurs de séquences JSON et une pour les analyseurs. En application du principe de robustesse, la grammaire est plus tolérante pour les seconds : elle permet que les membres d'une séquence ne soient pas des textes JSON valides, afin que l'analyseur puisse continuer à traiter la séquence (cf. section 2.3). En revanche, le générateur **doit** produire une séquence composée uniquement de textes JSON valides. La grammaire stricte, pour les générateurs, est donc :

1. Pour voir le RFC de numéro NNN, `https://www.ietf.org/rfc/rfcNNN.txt`, par exemple `https://www.ietf.org/rfc/rfc8259.txt`

```
JSON-sequence = *(RS JSON-text LF)

RS = %x1E
LF = %x0A

JSON-text = <given by RFC7159, using UTF-8 encoding>
```

RS et LF sont définis dans le RFC 20. À noter que RS, caractère ASCII très peu utilisé, se nomme "*Record Separator*" en ASCII mais "*INFORMATION SEPARATOR TWO*" dans Unicode. Le choix de ce séparateur a suscité d'intenses débats dans le groupe de travail, avant que le consensus se fasse sur RS. En langue naturelle, cette grammaire se résume en « une séquence JSON est composée d'un nombre quelconque de textes JSON, chacun précédé d'un RS et suivi d'un LF ». Ils sont forcément encodés en UTF-8 (RFC 3629). Comme RS est un caractère de contrôle, il ne peut pas apparaitre directement dans un texte JSON (RFC 8259, section 7) et il n'y a donc pas de risque de collision avec un vrai RS. Je laisse mes lecteurs aventureux chercher comment on entre un caractère RS dans un fichier, avec leur éditeur favori...

Autre grammaire, plus laxiste, pour les analyseurs. Elle est :

```
JSON-sequence = *(1*RS possible-JSON)
...
possible-JSON = 1*(not-RS)
not-RS = %x00-1d / %x1f-ff; any octets other than RS
```

Elle est bien plus tolérante que la grammaire du générateur. L'idée est que, si l'analyse d'un texte « possiblement JSON » échoue, l'analyseur pourra sauter au suivant (éventuellement en signalant à l'application qu'il y a eu un problème) et traiter le reste de la séquence. Comme le décrit bien la section 2.3, une erreur n'est pas forcément fatale. Si l'analyseur était trop puriste, il ne pourrait pas traiter un journal où certaines entrées ont été tronquées (et ne sont donc plus des textes JSON valides) suite à, par exemple, un disque plein.

Ce principe de robustesse ne pose pas de problèmes si les textes JSON sont des tableaux ou des objets : une éventuelle troncation se détecte sans ambiguité. Ainsi, `[116, 943, 234, 3879` a clairement été tronqué (il manque le crochet final). Il y a davantage de problèmes dans les cas où les textes JSON sont des entiers ou des littéraux comme `true`. Si on trouve `3879`, était-ce bien 3879 ou bien la troncation d'un entier plus long ? C'est là que le LF (U+000A) à la fin de chaque texte JSON est utile, comme canari pour détecter une troncation. Un analyseur doit donc vérifier sa présence (le RFC est plus tolérant, en acceptant n'importe quel `ws` - RFC 8259, section 2) si le texte JSON n'était pas auto-délimité (les tableaux, objets et chaînes sont auto-délimités, mais pas les nombres ou certains littéraux comme `null`).

Au passage, un mot sur la sécurité (section 3) : les analyseurs de séquences JSON, comme tous les analyseurs de JSON, seront souvent utilisés sur des documents venus de l'extérieur, pas forcément validés. Ils doivent donc être robustes et ne pas faire de "*buffer overflow*" sous prétexte qu'ils rencontrent du JSON bizarre. Et, si vous voulez signer vos séquences JSON, n'oubliez pas que JSON n'a pas de forme canonique, et les séquences encore moins (comme l'analyseur est plus laxiste que le générateur, lire et écrire une séquence peut la changer, par exemple en ajoutant des LF à la fin des textes). Toute opération peut donc potentiellement invalider une signature.

Question mises en œuvre, il semble (je n'ai pas encore testé) que le couteau suisse de JSON, jq `<https://www.bortzmeyer.org/jq.html>`, gère ce nouveau format.

Notez un format proche (mais différent : ce serait trop simple autrement), JSON Lines `<http://jsonlines.org/>`.