RFC 9535 : JSONPath: Query Expressions for JSON

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 14 février 2025

Date de publication du RFC : Février 2024

https://www.bortzmeyer.org/9535.html

Le langage JSONPath sert à désigner une partie d'un document JSON. Il est donc utile pour, par exemple, extraire des valeurs qui nous intéressent.

Il existe de nombreux langages pour cela (une liste partielle figure à la fin de l'article), pas forcément normalisés mais mis en œuvre dans tel ou tel logiciel (dont, évidemment, jq <https://www.bortzmeyer.org/jq.html>). Ce RFC est, je crois, la première normalisation formelle d'un tel langage. Voici tout de suite un exemple, utilisant le logiciel jpq <https://codeberg.org/KMK/jpq>, sur la la base des arbres parisiens <https://www.data.gouv.fr/fr/datasets/les-arbres-paris/> :

```
% jpq '$[*].libellefrancais' arbres.json
[
  "Marronnier",
  "Aubépine",
  "Cerisier à fleurs",
...
```

Cela se lit « affiche les noms d'espèce [`libellefrancais`, dans le fichier] de tous [vous avez vu l'astérisque] les arbres ». (Avec jq, on aurait écrit `jq '.[].libellefrancais' arbres.json`.) Voyons maintenant les détails.

JSONPath travaille sur des documents JSON, JSON étant normalisé dans le RFC 8259 [1]. Il n'est sans doute pas utile de présenter JSON, qui est utilisé partout aujourd'hui. Notons simplement que l'argument d'une requête JSONPath, écrit en JSON, est ensuite modélisé comme un arbre, composé de nœuds. Le JSON {`"foo": 3, "bar": 0`} a ainsi une racine de l'arbre sous laquelle se trouvent deux nœuds, avec comme valeur 3 et 0 (les noms des membres JSON, `foo` et `bar`, ne sont pas sélectionnables par JSONPath et donc pas considérés comme nœuds).

Vous êtes impatient·e de vous lancer ? On va utiliser le logiciel jpq <https://codeberg.org/KMK/jpq> cité plus haut. Pensez à installer un Rust **très** récent puis :

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

```
cargo install jpq
```

Ou bien, si vous voulez travailler à partir des sources :

```
git clone https://codeberg.org/KMK/jpq.git
cd jpq
cargo install --path .
```

Puis pensez à modifier votre chemin de recherche d'exécutables si ce n'est pas déjà fait (notez que l'excellente Julia Evans vient justement de faire un très bon article sur ce sujet <https://jvns.ca/blog/2025/02/13/how-to-add-a-directory-to-your-path/>). Testez :

```
% echo '["bof"]' | jpq '$[0]'
[
  "bof"
]
```

Les choses utiles à savoir pour écrire des expressins JSONPath :

— Le dollar indique la racine de l'arbre JSON.
— L'expression JSONPath est faite de **segments**, chaque segment permettant de filtrer une partie du document JSON (dans l'exemple immédiatement au-dessus, `[0]` est un segment sélectionnant le premier élément d'un tableau, ici, le seul).
— Des **sélecteurs** sont utilisés dans les segments pour sélectionner via le nom d'un membre JSON ou via un indice (0 dans l'exemple plus haut).

La grammaire complète (en ABNF, RFC 5234) figure dans l'annexe A du RFC.

Allez, passons aux exemples, avec les arbres parisiens <https://www.data.gouv.fr/fr/datasets/les-arbres-paris/> comme document JSON. Le quatrième arbre du document (on part de zéro) :

```
% jpq '$[3]' arbres.json
[
  {
    "idbase": 143057,
    "typeemplacement": "Arbre",
    "domanialite": "CIMETIERE",
    "arrondissement": "PARIS 20E ARRDT",
    "complementadresse": null,
    "numero": null,
    "adresse": "CIMETIERE DU PERE LACHAISE / DIV 41",
    "idemplacement": "D00000041050",
    "libellefrancais": "Erable",
...
```

Tous les genres (dans la classification de Linné) :