RFC 9165 : Additional Control Operators for Concise Data Definition Language (CDDL)

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 25 décembre 2021

Date de publication du RFC : Décembre 2021

https://www.bortzmeyer.org/9165.html

Le langage CDDL ("*Concise Data Definition Language*") est un langage de description de schémas de données, notamment pour le format CBOR. Ce nouveau RFC étend CDDL avec de nouveaux opérateurs, permettant entre autres l'addition d'entiers et la concaténation de chaines de caractères.

CDDL est normalisé dans le RFC 8610[1] (et le format CBOR dans le RFC 8949). Il permet l'ajout de nouveaux opérateurs pour étendre le langage, possibilité utilisée par notre nouveau RFC. Notez que, comme les modèles de données de JSON et CBOR sont très proches, les schémas CDDL peuvent également être utilisés pour JSON, ce que je fais ici pour les exemples car le JSON est plus facile à lire et à écrire.

D'abord, l'opérateur `.plus`. Il permet par exemple, dans la spécification d'un schéma, de faire dépendre certains nombres d'autres nombres. L'exemple ci-dessous définit un type « intervalle » où la borne supérieure doit être supérieure de 5 à la borne inférieure :

```
top = interval<3>

interval<BASE> = BASE .. (BASE .plus 5)
```

Avec un tel schéma, la valeur 4 sera acceptée mais 9 sera refusée :

1. Pour voir le RFC de numéro NNN, `https://www.ietf.org/rfc/rfcNNN.txt`, par exemple `https://www.ietf.org/rfc/rfc8610.txt`

```
% cddl tmp.cddl validate tmp.json
CDDL validation failure (nil for 9):
[9, [:range, 3..8, Integer], ""]
```

Deuxième opérateur, la concaténation, avec le nouvel opérateur `.cat` :

```
s = "foo" .cat "bar"
```

Dans cet exemple, évidemment, `.cat` n'est pas très utile, on aurait pu écrire la chaine complète directement. Mais `.cat` est plus pertinent quand on veut manipuler des chaines contenants des sauts de ligne :

```
s = "foo" .cat '
 bar
 baz
'
```

Ce schéma acceptera la chaine de caractère `"foo\n bar\n baz\n"`.

Dans l'exemple ci-dessus, `bar` et `baz` seront précédés des espaces qui apparaissaient dans le code source. Souvent, on souhaite mettre ces espaces en début de ligne dans le code source, pour l'indenter joliment, mais les supprimer dans le résultat final. Cela peut se faire avec l'opérateur pour lequel notre RFC invente le joli mot de détentation (*"dedending"*), `.det`, qui fonctionne comme `.cat` mais « dédente » les lignes :

```
s = "foo" .det '
 bar
 baz
'
```

Cette fois, le schéma n'acceptera que la chaine `"foo\nbar\nbaz\n"`.

Le RFC note que, comme `.det` est l'abréviation de *"dedending cat"*, on aurait pu l'appeler `.dedcat` mais cela aurait chagriné les amis des chats.

CDDL est souvent utilisé dans les normes techniques de l'Internet et celles-ci contiennent souvent des grammaires en ABNF (RFC 5234). Pour permettre de réutiliser les règles ABNF dans CDDL, et donc se dispenser d'une ennuyeuse traduction, un nouvel opérateur fait son apparition, `.abnf`. Le RFC donne l'exemple de la grammaire du RFC 3339, qui normalise les formats de date : (en ligne sur `https://www.bortzmeyer.org/files/abnf-rfc3339.cddl`). Avec ce fichier, on peut accepter des chaines comme `"2021-12-15"` ou `"2021-12-15T15:52:00Z"`. Notons qu'il reste quelques difficultés car les règles d'ABNF ne sont pas parfaitement compatibles avec celles de CDDL. Si `.abnf` va traiter l'ABNF comme de l'Unicode encodé en UTF-8, un autre opérateur, `.abnfb`, va le traiter comme une bête suite d'octets. D'autre part, comme ABNF exige souvent des sauts de ligne, les opérateurs `.cat` et `.det` vont être très utiles.

Quatrième et dernier opérateur introduit par ce RFC, `.feature`. À quoi sert-il ? Comme le langage CDDL peut [Caractère Unicode non montré [2]]tre étendu au-delà de ce qui existait dans le RFC 8610, on court toujours le risque de traiter un schéma CDDL avec une mise en œuvre de CDDL qui ne connait pas toutes les fonctions utilisées dans le schéma. Cela produit en général un message d'erreur peu clair et, surtout, cela mènerait à considérer des données comme invalides alors qu'elles sont parfaitement acceptables pour le reste du schéma. `.feature` sert donc à marquer les extensions qu'on utilise. Le programme qui met en œuvre CDDL pourra ainsi afficher de l'information claire. Par exemple, si on définit une personne :

```
person = {
     ? name: text
     ? organization: text
}
```

puis qu'on veut rajouter son groupe sanguin :

```
{"name": "Jean", "bloodgroup": "O+"}
```

Cet objet sera rejeté, en raison du champ `bloodgroup`. On va faire un schéma plus ouvert, avec `.feature` :

```
person = {
     ? name: text
     ? organization: text
     * (text .feature "further-person-extension") => any
}
```

Et, cette fois, l'objet est accepté avec un message d'avertissement clair :

```
% cddl person-new-feature.cddl validate tmp.json
** Features potentially used (tmp.json): further-person-extension: ["bloodgroup"]
```

Comme le schéma est assez ouvert, la fonction de génération de fichiers d'exemple de l'outil donne des résultats amusants :

```
% cddl person-new-feature.cddl generate
{"name": "plain", "dependency's": "Kathryn's", "marvelous": "cleavers"}
```

Les nouveaux opérateurs ont été placés dans le registre IANA <`https://www.iana.org/assignments/cddl/cddl.xml#cddl-control-operators`>. Ils sont mis en œuvre dans l'outil de référence de CDDL (le `cddl` utilisé ici). Écrit en Ruby, on peut l'installer avec la méthode Ruby classique :

2. Car trop difficile à faire afficher par LaTeX