RFC 7095 : jCard: The JSON format for vCard

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 15 janvier 2014

Date de publication du RFC : Janvier 2014

http://www.bortzmeyer.org/7095.html

Le format vCard permet de décrire de manière standard, compréhensible par tous les logiciels de gestion de carnet d'adresses, les informations sur un individu ou une organisation. Très connu et largement déployé, il est présent à de nombreux endroits. vCard est à la fois un modèle de données (décrivant ce qu'on stocke) et une syntaxe particulière. Des syntaxes alternatives sont possibles et ce nouveau RFC en propose une, fondée sur JSON. Elle se nomme donc **jCard**.

Il y avait déjà une autre syntaxe alternative, fondée sur XML, dans le RFC 6351 [1]. Celle-ci s'inscrit dans la mode JSON actuelle et promet de rendre le vieux format vCard plus accessible pour les applications Web, notamment celles écrites en JavaScript.

vCard est actuellement en version 4, normalisée dans le RFC 6350. Il a plusieurs points techniques en commun avec le format iCalendar de gestion d'agendas, normalisé dans le RFC 5545. Le travail portait donc également sur une syntaxe JSON pour iCalendar, normalisée dans le RFC 7265. JSON, lui, est normalisé dans le RFC 8259. La section 1 du RFC résume le cahier des charges du projet jCard :

— Possibilité de conversion vCard -¿ jCard et retour, sans perte sémantique,
— Pas d'impératif de préservation de l'ordre des éléments,
— Maintien du modèle de données de vCard,
— Les futures extensions de vCard devront pouvoir être intégrées naturellement, sans nécessiter un nouveau RFC jCard.

La section 3 est le gros morceau du RFC, indiquant comment convertir les vCard actuelles en jCard. Rappelons qu'une vCard comprend plusieurs propriétés (de cardinalité pouvant être différente de 1), chacune ayant une valeur et, parfois, des paramètres, ces derniers ayant un nom et une valeur. Par exemple, dans :

1. Pour voir le RFC de numéro NNN, https://www.ietf.org/rfc/rfcNNN.txt, par exemple https://www.ietf.org/rfc/rfc6351.txt

```
TEL;TYPE=VOICE,TEXT,WORK;VALUE=uri:tel:+1-666-555-1212
```

La propriété est `TEL`, il y a deux paramètres, `TYPE` qui a trois valeurs, et `VALUE` qui n'en a qu'une. La valeur de la propriété est un numéro de téléphone, ici +1-666-555-1212.

vCard a quelques particularités syntaxiques. Par exemple, les lignes trop longues peuvent être repliées, et certaines valeurs ont besoin d'échappement. Pour passer à jCard, on déplie toutes les lignes et on retire les échappements. Puis on applique les échappements de JSON.

Une vCard est représentée par un tableau JSON de deux éléments, le premier étant juste la chaîne de caractères `vcard` et le second étant un tableau JSON avec un élément par propriété. Si on veut mettre plusieurs vCard ensemble, on peut faire un tableau JSON dont chaque élément est une vCard représentée en JSON comme ci-dessus.

Chaque élément qui représente une propriété se compose d'un tableau d'au moins quatre éléments, les quatre obligatoires étant le nom de la propriété (forcément en minuscule, alors que cela ne comptait pas en vCard), les paramètres, une chaîne identifiant le type de la valeur, et la valeur. Si la propriété avait plusieurs valeurs, ce qui est permis en vCard, on ajoute des éléments au tableau. Un exemple de jCard est donc :

```
["vcard",
 [
    ["version", {}, "text", "4.0"],
    ["fn", {}, "text", "John Doe"],
    ["gender", {}, "text", "M"],
    ["categories", {}, "text", "computers", "cameras"],
    ...
 ]
]
```

On voit ici une propriété `gender` (genre, normalisée dans la section 6.2.7 du RFC 6350), et ayant une valeur de type texte. La propriété `categories` (section 6.7.1 du RFC 6350), elle, a deux valeurs. On notera que, dans cet exemple, aucune propriété n'a de paramètre, ce qui est représenté par l'objet JSON vide {}. On notera aussi qu'un seul type est utilisé, `text` mais on verra plus loin quelques exemples de la riche liste de types possibles (le type doit être indiqué explicitement en jCard pour permettre à jCard de traiter des futures extensions à vCard, sans avoir besoin de connaître les nouvelles propriétés ; voir la section 5 pour les détails).

Quelques propriétés sont traitées de manière particulière. Ainsi, `version` (4.0 à l'heure actuelle) doit être la première du tableau. Certaines propriétés, en vCard, sont structurées : l'exemple le plus classique est l'adresse `ADR`, avec des champs séparés par un point-virgule. En jCard, on les représente par un tableau. Ainsi, la propriété vCard :

```
ADR:;;123 Main Street;Any Town;CA;91921-1234;U.S.A.
```

devient en jCard :

```
["adr", {}, "text",
     [
     "", "", "123 Main Street",
     "Any Town", "CA", "91921-1234", "U.S.A."
     ]
   ]
```

(Notez les deux champs manquants au début, le numéro de la boîte postale, s'il y en a une, et le numéro de l'appartement.)

Et les paramètres ? C'est un objet JSON (ce qu'on nomme un dictionnaire dans d'autres langages). Chaque membre de l'objet a pour clé un nom de paramètre et pour valeur la valeur du paramètre. Si l'adresse ci-dessus avait eu le paramètre vCard `LANGUAGE` avec une valeur indiquant une adresse rédigée en anglais, la première ligne du jCard deviendrait :

```
["adr", {"language": "en"}, "text",
 ...
```

On a fait les propriétés, les paramètres, restent les valeurs. Il existe de nombreux types possibles pour les valeurs, je ne vais pas les citer tous. Commençons par l'exemple `uri`. Une valeur de ce type contient... un URI :

```
["photo", {}, "uri", "http://www.bortzmeyer.org/images/moi.jpg"]
```

Les dates n'ont pas tout à fait le même format qu'en vCard. En jCard, elles suivent la norme ISO 8601 2004 donc, par exemple :

```
["bday", {}, "date", "1412-01-06"]
```

(au lieu du `BDAY:14120106` de vCard.)

Autre exemple de type utile, `language-tag`, pour indiquer que la valeur est une étiquette de langue :

```
["lang", {}, "language-tag", "de"]
```

(Petit rappel vCard au passage : `language` est la langue dans laquelle est écrite un élément d'information, par exemple une adresse. `lang` est la langue qu'on peut utiliser pour contacter l'entité décrite par une vCard.)

La section 4 explique, elle, les détails de l'opération inverse, convertir du jCard en vCard.

Reste le problème des propriétés ou paramètres inconnus (section 5). Si un nouveau RFC crée des extensions à vCard, par exemple en ajoutant des propriétés, comme le RFC 6474 ajoutait `BIRTHPLACE`, `DEATHPLACE` et `DEATHDATE`, que faire ? La section 5 étudie leur représentation en jCard. Rappelez-vous qu'un des éléments du cahier des charges était la possibilité d'étendre vCard sans avoir besoin de créer un nouveau RFC pour jCard. Lorsqu'un convertisseur vCard -¿ jCard rencontre une propriété de type inconnu, il peut mettre `unknown` comme type. Ainsi, `FOO:http://www.foo.example/` deviendra :

```
["foo", {}, "unknown", "http://www.foo.example/"]
```

alors qu'un convertisseur plus récent, connaissant la propriété (imaginaire, hein, c'est juste un exemple) `FOO` pourra traduire :

```
["foo", {}, "uri", "http://www.foo.example/"]
```

Ce nouveau format jCard a désormais son propre type MIME, `application/vcard+json` (et pas `jcard+json`, attention).

Parmi les mises en œuvre de jCard, on note :

— ICAL.js <`https://github.com/mozilla-comm/ical.js/`> en JavaScript (mais surtout prévu pour iCalendar en JSON),

— Py Calendar <`https://svn.calendarserver.org/repository/calendarserver/PyCalendar/branches/json/`>, en Python, également surtout pour iCalendar,

— ez-vcard <`https://code.google.com/p/ez-vcard/`>, en Java. Ne compile pas du premier coup (`mvn install` -¿ *"Tests in error : HCardPageTest.create_then_parse :259 NullPointer"*) mais semble bien conçu, généralement.

Voici ma vCard présentée dans mon article sur le RFC 6350, traduite en jCard (un exemple complet se trouve sinon dans l'annexe B.1 du RFC) :

```
["vcard",
  [
     ["version", {}, "text", "4.0"],
     ["fn", {}, "text", "Stéphane Bortzmeyer"],
     ["n", {}, "text", ["Bortzmeyer", "Stéphane", "", ""]],
     ["uid", {}, "uri", "urn:uuid:a06340f8-9aca-41b8-bf7a-094cbb33d57e"],
     ["gender", {}, "text", "M"],
     ["kind", {}, "text", "individual"],
     ["email", {}, "text", "individual"],
     ["title", {}, "text", "Indigène de l'Internet"],
     ["photo", {}, "uri", "http://www.bortzmeyer.org/images/moi.jpg"],
     ["lang", {"pref": "1"}, "language-tag", "fr"],
     ["lang", {"pref": "2"}, "language-tag", "en"],
     ["impp", {"pref": "1"}, "uri", "xmpp:bortzmeyer@dns-oarc.net"],
     ["impp", {"pref": "2"}, "uri", "xmpp:bortzmeyer@gmail.com"],
     ["url", {}, "uri", "http://www.bortzmeyer.org/"],
     ["key", {}, "uri", "http://www.bortzmeyer.org/files/pgp-key.asc"]
  ]
]
```