RFC 6352 : vCard Extensions to WebDAV (CardDAV)

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 1 septembre 2011. Dernière mise à jour le 18 août 2014

Date de publication du RFC : Août 2011

https://www.bortzmeyer.org/6352.html

De même que le format de fichiers iCal, décrivant des événements, avait son protocole CalDAV pour synchroniser des agendas entre machines, le format vCard décrivant des informations sur une personne a désormais son protocole CardDAV, pour synchroniser des carnets d'adresses.

Lorsque ce RFC sera largement déployé, on pourra donc sauvegarder, distribuer, partager son carnet d'adresses de manière standard. Cela se fait en définissant, comme pour CalDAV, des **extensions** au protocole WebDAV (RFC 4918 [1]).

Ce n'est pas qu'on manquait de protocoles pour l'accès à un carnet. Comme le rappelle la section 1, on avait LDAP (RFC 4510), IMSP et ACAP (RFC 2244), ainsi que SyncML. Mais bâtir sur WebDAV (RFC 4918) permettait de récupérer les fonctions perfectionnées de ce protocole. Le nouveau CardDAV permet d'avoir plusieurs carnets d'adresses, d'en protéger l'accès par des autorisations (RFC 3744), de faire des recherches côté serveur (sans avoir besoin de récupérer tout le carnet d'adresses chez le client), etc. WebDAV étant lui-même bâti sur HTTP, on récupère aussi les fonctions de HTTP comme la sécurité avec TLS (RFC 2818). CardDAV utilise le format vCard. Ce format avait été normalisé dans le RFC 2426 et se trouve désormais dans la RFC 6350. CardDAV dépend de certaines extensions à WebDAV et tout serveur WebDAV ne conviendra pas forcément. La section 3 résume les exigences de CardDAV (serveur permettant de créer des collections avec `MKCOL` - RFC 5689 et exemple d'usage en section 6.3.1, versionnement - RFC 3253, etc). Par contre, il n'y a pas encore dans le protocole CardDAV de mécanisme de notification asynchrone (≪ Machin a changé de numéro de téléphone ≫). Le RFC ne le précise pas mais il y a un inconvénient à utiliser WebDAV, la complexité. Notre RFC 6352 fait 56 pages, et il faut avoir compris WebDAV avant...

1. Pour voir le RFC de numéro NNN, `https://www.ietf.org/rfc/rfcNNN.txt`, par exemple `https://www.ietf.org/rfc/rfc4918.txt`

La section 4 du RFC explique le modèle de données utilisé pour le carnet : chaque carnet d'adresses est une **collection** WebDAV et chaque entrée dans le carnet est une ressource WebDAV, adressable et verrouillable séparement. En même temps que les carnets d'adresses, un serveur CardDAV peut aussi gérer avec WebDAV d'autres ressources (un exemple serait un serveur où le carnet d'adresses de Lisa serait en `/addressbooks/lisa` et son agenda, via CalDAV, en `/calendars/lisa`). Le serveur publie le fait qu'il gère CardDAV en ajoutant `addressbook` à la réponse à la requête HTTP `OPTIONS`.

La section 5 passe ensuite à ces ressources. Chacune d'elles est une entrée du carnet, typiquement exprimée en format vCard (le serveur peut gérer également d'autres formats).

Emprunté au RFC (section 6.1), voici un exemple de connexion à un serveur CardDAV :

```
(Le client demande)
OPTIONS /addressbooks/ HTTP/1.1
Host: addressbook.example.com

(Le serveur répond)
HTTP/1.1 200 OK
Allow: OPTIONS, GET, HEAD, POST, PUT, DELETE, TRACE, COPY, MOVE
Allow: MKCOL, PROPFIND, PROPPATCH, LOCK, UNLOCK, REPORT, ACL
DAV: 1, 2, 3, access-control, addressbook
DAV: extended-mkcol
Date: Sat, 11 Nov 2006 09:32:12 GMT
Content-Length: 0
```

Le client sait désormais que le serveur gère CardDAV (le `addressbook` dans l'en-tête `DAV`).

Différentes **propriétés** WebDAV (RFC 4918, section 4 ; les propriétés sont les métadonnées de WebDAV) permettent de se renseigner sur les carnets d'adresses. Elles sont formatées en XML. Ici, `addressbook-descri` (section 6.2.1 ; son nom indique bien à quoi elle sert) peut valoir, par exemple (notez l'attribut `xml:lang` qui illustre les capacités d'internationalisation de CardDav) :

```
<carddav:addressbook-description xml:lang="fr-CA"
   xmlns:carddav="urn:ietf:params:xml:ns:carddav">
  Adresses de Oliver Daboo
</carddav:addressbook-description>
```

Et pour créer des entrées dans le carnet d'adresses, ce qui est quand même la tâche la plus importante de CardDAV ? CardDAV étant fondé sur HTTP, on utilise évidemment une requête `PUT` (de même que la lecture se ferait par un `GET`). La section 6.3.2 nous fournit un exemple, avec l'entrée au format vCard :

```
PUT /lisa/addressbook/newvcard.vcf HTTP/1.1
If-None-Match: *
Host: addressbook.example.com
Content-Type: text/vcard
Content-Length: xxx

BEGIN:VCARD
VERSION:3.0
FN:Cyrus Daboo
N:Daboo;Cyrus
```

```
ADR;TYPE=POSTAL:;2822 Email HQ;Suite 2821;RFCVille;PA;15213;USA
EMAIL;TYPE=INTERNET,PREF:cyrus@example.com
NICKNAME:me
NOTE:Example VCard.
ORG:Self Employed
TEL;TYPE=WORK,VOICE:412 605 0499
TEL;TYPE=FAX:412 605 0705
URL:http://www.example.com
UID:1234-5678-9000-1
END:VCARD
```

Le `If-None-Match` dans la requête (RFC 7232, section 3.2) est là pour garantir qu'une ressource du même nom n'existe pas déjà. Cela évite d'effacer une ressource existante. Le `.vcf` à la fin de l'URL est l'extension commune des fichiers vCard.

Les carnets d'adresses sont évidemment des choses assez personnelles, on a donc des moyens de les protéger (section 7). Un serveur CardDAV permet de définir des ACL, comme normalisé dans le RFC 3744.

Récupérer un carnet d'adresses avec `GET` est simple mais ne laisse guère de choix. La section 8 présente des mécanismes plus sophistiqués. CardDAV a la méthode HTTP `REPORT` du RFC 3253 pour produire des extractions du carnet. Mais il a aussi des fonctions de recherche et de tri. Celles-ci posent le problème des règles de comparaison (rappelez-vous qu'un fichier vCard est de l'Unicode et peut contenir des caractères de toute sorte). La section 8.3 précise donc :

— Ces règles sont exprimées en utilisant la terminologie du RFC 4790, qui fournit une liste de règles `<https://www.iana.org/assignments/collation/collation-index.html>`,
— Parmi ces règles, le serveur doit accepter au moins les comparaisons ASCII `i;ascii-casemap` du RFC 4790 et les comparaisons Unicode `i;unicode-casemap` du RFC 5051,
— Les opérations possibles sont au minimum le test d'égalité et le test de sous-chaîne (« strauss » doit trouver « strauss-kahn »), avec ses options « au début de la chaîne » et « à la fin de la chaîne ».

Voici un exemple de commande `REPORT` en ajoutant une condition pour ne sélectionner que les entrées du carnet dont le `NICKNAME` est exactement égal à « me » :

La requête :

```
REPORT /home/bernard/addressbook/ HTTP/1.1
Host: addressbook.example.com
Depth: 1
Content-Type: text/xml; charset="utf-8"
Content-Length: xxxx

<?xml version="1.0" encoding="utf-8" ?>
<carddav:addressbook-query xmlns:dav="DAV:"
                 xmlns:carddav="urn:ietf:params:xml:ns:carddav">
  <dav:prop>
    <dav:getetag/>
    <carddav:address-data>
      <carddav:prop name="VERSION"/>
      <carddav:prop name="UID"/>
      <carddav:prop name="NICKNAME"/>
      <carddav:prop name="EMAIL"/>
      <carddav:prop name="FN"/>
    </carddav:address-data>
  </dav:prop>
  <carddav:filter>
    <carddav:prop-filter name="NICKNAME">
      <carddav:text-match collation="i;unicode-casemap"
                   match-type="equals">
```

```
            me
          </carddav:text-match>
        </carddav:prop-filter>
      </carddav:filter>
    </carddav:addressbook-query>

La réponse :

    HTTP/1.1 207 Multi-Status
    Date: Sat, 11 Nov 2006 09:32:12 GMT
    Content-Type: text/xml; charset="utf-8"
    Content-Length: xxxx

    <?xml version="1.0" encoding="utf-8" ?>
    <dav:multistatus xmlns:dav="DAV:"
                     xmlns:carddav="urn:ietf:params:xml:ns:carddav">
      <dav:response>
        <dav:href>/home/bernard/addressbook/v102.vcf</dav:href>
        <dav:propstat>
          <dav:prop>
            <dav:getetag>"23ba4d-ff11fb"</dav:getetag>
            <carddav:address-data>BEGIN:VCARD
    VERSION:3.0
    NICKNAME:me
    UID:34222-232@example.com
    FN:Cyrus Daboo
    EMAIL:daboo@example.com
    END:VCARD
    </carddav:address-data>
          </dav:prop>
          <dav:status>HTTP/1.1 200 OK</dav:status>
        </dav:propstat>
      </dav:response>
    </dav:multistatus>
```

Écrire un client CardDAV n'est pas trivial, pas seulement à cause de la longueur du RFC mais aussi parce qu'il existe plusieurs pièges. La lecture de la section 9 est donc recommandée. Elle rappelle par exemple que le client peut ne demander qu'une **partie** des données (dans l'exemple ci-dessus, le client demande à ne voir que `VERSION`, `UID`, `NICKNAME`, `EMAIL` et `FN`). Cela lui évite de devoir télécharger des données qui peuvent être de grande taille comme `PHOTO` ou `SOUND`. Par contre, cette astuce ne marche qu'en lecture, pas en écriture : un `PUT` doit transmettre la totalité de la "*vCard*" et le protocole ne fournit pas de moyen de ne mettre à jour qu'un seul champ. Le client doit donc récupérer toute la "*vCard*", changer un champ et renvoyer toute la "*vCard*" modifiée.

Piège classique dans ce cas, la « mise à jour perdue ». Que se passe-t-il si les données ont été changées entre leur récupération et le renvoi ? Ce changement risque d'être écrasé. Pour éviter cela, le client a tout intérêt à utiliser l'option `If-Match:` de la requête HTTP, en indiquant comme paramètre un "*Etag*" récupéré précedemment (les "*Etags*" sont décrits dans la section 2.3 du RFC 7232). Ainsi, dans le cas cité ci-dessus, le renvoi du "*vCard*" modifié sera refusé, préservant la mise à jour qui avait été faite parallèlement. (Emmanuel Saint-James me fait remarquer qu'on aurait aussi pu utiliser `If-Unmodified-Since:`, si le serveur CardDAV met les informations dans une base de données qui stocke la date de modification, par exemple le système de fichiers Unix ; cette possibilité n'est pas mentionnée par le RFC.)

Comment configure-t-on un client CardDAV (section 9.3) ? On a vu que l'URL utilisé était arbitraire, au choix de l'implémenteur. Le client doit donc être configuré avec cet URL. Il existe bien deux mécanismes de découverte automatique, décrits dans le RFC 5397 et dans le RFC 6764, mais qui ne sont pas forcément présents partout. (iOS utilise celui du RFC 6764 donc si vous voyez des requêtes `/.well-known/carddav` passer, c'est lui.)

Et le nom du serveur ? La section 11 décrit un enregistrement SRV pour le trouver. Le nom `_carddav._tcp.domain.example` permet donc de trouver le serveur CardDAV du domain `domain.example` (et `_carddavs` pour utiliser TLS). Si l'enregistrement SRV vaut :

```
_carddav._tcp     SRV 0 1 80 addressbook.example.com.
```

Cela indique que ce serveur est `addressbook.example.com` et qu'il est accessible sur le port 80. Si ces enregistrements SRV sont présents, et si la propriété `current-user-principal-URL` du RFC 5397 est utilisée, il n'y a plus à configurer que le nom de domaine, le nom de l'utilisateur et un mot de passe.

Et si on veut accéder au carnet d'un autre utilisateur, ce qui peut arriver en cas de carnets partagés ? La section 9.4 rappelle l'existence de la propriété `principal-property-search` du RFC 3744, qui peut permettre de demander un `REPORT` en cherchant le carnet sur divers critères. Ici, on cherche le carnet de Laurie :

```
REPORT /home/bernard/addressbook/ HTTP/1.1
Host: addressbook.example.com
...
<dav:principal-property-search xmlns:dav="DAV:">
  <dav:property-search>
    <dav:prop>
      <dav:displayname/>
    </dav:prop>
    <dav:match>Laurie</dav:match>
  </dav:property-search>
  <dav:prop>
    <carddav:addressbook-home-set
       xmlns:carddav="urn:ietf:params:xml:ns:carddav"/>
    <dav:displayname/>
  </dav:prop>
</dav:principal-property-search>
```

Notez que, CardDAV s'appuyant sur XML, il hérite de ses capacités d'internationalisation (section 12), notamment l'usage d'Unicode.

Et question sécurité ? Des failles dans CardDAV ? La section 13 tente de prévenir les problèmes : d'abord, CardDAV hérite des questions de sécurité de HTTP. Si on n'utilise pas HTTPS, les requêtes et réponses circulant sur le réseau peuvent être lues et/ou modifiées. C'est particulièrement important si on utilise l'authentification HTTP de base, avec envoi d'un mot de passe.

Ensuite, une fois les utilisateurs authentifiés, encore faut-il éviter qu'un utilisateur ne tripote les carnets des autres. L'extension « ACL » du RFC 3744 permet d'atteindre cet objectif. Elle permet d'avoir des carnets publics, partagés ou complètement privés. Le serveur doit ensuite veiller au respect de cette classification.

Le schéma XML complet pour les éléments CardDAV figure en section 10. Question implémentations, on peut regarder la liste gérée par les gens de Zimbra <http://wiki.zimbra.com/wiki/CardDAV>, celle de Trinity <http://www.trinitydesktop.org/relatedprojects.php>, celle de Wikipédia <http://en.wikipedia.org/wiki/CardDAV#Implementation>. Parmi les programmes qui gèrent CardDAV, citons le serveur Davical, Radicale <http://radicale.org/> ou Evolution. le client Kontact de KDE <http://code.google.com/p/kcarddav/> n'est plus maintenu. Pour Firefox, c'est en discussion <https://bugzilla.mozilla.org/show_bug.cgi?id=859306>. Merci à Mathieu Dupuy pour les détails.