RFC 6690 : CoRE Link Format

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 16 août 2012

Date de publication du RFC : Août 2012

https://www.bortzmeyer.org/6690.html

Le contexte de ce nouveau RFC est celui des machines ayant peu de ressources, les environnements « contraints ». Les serveurs HTTP tournant sur ces machines ont besoin d'un moyen de faire connaître la liste des services qu'ils proposent. Ce RFC propose de le faire en distribuant en REST des fichiers d'un format spécifique (normalisé ici) listant ces services. Le format s'appuie sur les liens du RFC 8288 [1].

Le groupe de travail IETF CoRE <http://tools.ietf.org/wg/core> travaille pour ces pauvres machines contraintes, n'ayant parfois qu'un processeur 8-bits et peu de mémoire, connectées par des LowPAN (RFC 4919). Il se focalise sur les applications de machine à machine, où un programme parle à un autre (et pas à un humain, comme la plupart des serveurs Web).

Le problème spécifique traité par ce RFC est celui de la **découverte** des services (typiquement représentés par un URI) offerts par une machine. Il n'y a pas d'humain dans la boucle, pour lire un mode d'emploi ou une documentation. Et des interfaces statiques sont trop rigides. L'idée de ce RFC est donc de mettre ces URI sous un format standard, dans une ressource à l'URI bien connu, `/.well-known/core` (les URI `.well-known` sont normalisés dans le RFC 8615). La ressource en question (cela peut être un fichier statique ou bien elle peut être construite à la demande) est récupérée en HTTP (`GET /.well-known/core`) ou bien avec le protocole COAP (RFC 7252). Et le format est une concrétisation du concept de lien décrit dans le RFC 8288.

Ainsi, un client CoRE pourra récupérer la liste des variables mesurées par un capteur, par exemple. C'est évidemment très limité par rapport à un moteur de recherche, mais c'est implémentable dans des machines même très limitées.

1. Pour voir le RFC de numéro NNN, `https://www.ietf.org/rfc/rfcNNN.txt`, par exemple `https://www.ietf.org/rfc/rfc8288.txt`

Le RFC 8288 décrivait un concept abstrait de **lien**, pour lequel il peut y avoir plusieurs sérialisations (liens `<a>` de HTML, en-têtes HTTP, liens Atom, etc). La sérialisation normalisée ici est conçue pour être compacte et triviale à analyser (rappelez-vous : machines contraintes), mais les mêmes données peuvent également être servies par des protocoles et des formats plus classiques, si on a aussi des clients plus puissants. Le format compact de ce RFC est identifié par le type `application/link-format`.

Chaque URI indique un service hébergé (section 2 du RFC). On les écrit entre chevrons. On peut les voir comme un lien « héberge » entre le serveur et le service (« ce serveur héberge - *"hosts"* - ce service »).

Voici un exemple de liens suivant ce format, adapté de la section 5 du RFC. Le serveur permet de contrôler des capteurs de température et de lumière. Il y a donc deux services :

```
</sensors/temp>;if="sensor",</sensors/light>;if="sensor"
```

Désolé de ne pas pouvoir vous montrer d'exemple réel mais une recherche Google `inurl:".well-known/core"` ne donne pas le résultat attendu. Trop malin, Google décide d'ignorer ce qu'il prend pour de la ponctuation.

Les URI indiqués sont relatifs. Ils sont relatifs à un **contexte** qui est en général l'endroit où on a obtenu la ressource `/.well-known/core`.

Ces liens, comme vous avez vu dans les exemples plus haut, peuvent contenir des attributs (section 3 du RFC). Les attributs spécifiques à ce format sont :

- `rt` (*"Resource type"*), qui identifie le service (c'est un identificateur formel, pas un texte lisible pour les humains). Pour prendre l'exemple d'un service de mesure de la température, `rt` peut être une simple chaîne de caractères comme `outdoor-temperature` ou bien un URI pointant vers une ontologie comme `http://sweet.jpl.nasa.gov/2.0/phys.owl#Temperature`.
- `if` (*"Interface"*), qui indique l'URI par lequel on va interagir avec le service. Plusieurs services peuvent avoir la même interface (comme dans le premier exemple donné plus haut avec le capteur de température et celui de lumière). La valeur de `if` peut être un URI pointant vers une description en WADL.
- `sz` (*"Size"*), qui indique la taille maximale de la ressource qu'on va récupérer, une information essentielle pour les machines contraintes ! Cette information peut être omise si elle est inférieure à la MTU. Un exemple serait `</firmware/v2.1>;rt="firmware";sz=262144`.

Les attributs standards des liens peuvent aussi être utilisés, comme `title` (RFC 8288, section 5.4).

Voici pour la syntaxe du fichier (de la ressource) récupérée. Et pour le récupérer, ce fichier (cette ressource) ? La section 4 revient en détail sur le concept d'URI bien connu (normalisé dans le RFC 5785) et sur celui utilisé par CoRE, `/.well-known/core`. Un point intéressant est la possibilité de filtrer les requêtes (section 4.1). La syntaxe est définie par un gabarit RFC 6570 : `/.well-known/core{?search*}`. ce qui permet des requêtes comme :

- `?href=/foo*`, pour se limiter aux liens commençant par `/foo`,
- `?foo=bar` pour se limiter aux liens ayant un attribut `foo` dont la valeur est `bar`,
- `?foo=*` pour ne garder que les liens ayant un attribut `foo` (valeur quelconque).

Le serveur étant une machine contrainte, il a parfaitement le droit de ne pas mettre en œuvre les filtres (dans la libcoap, présentée plus loin, il y a une option de compilation `--without-query-filter` pour supprimer les filtres et réduire ainsi la taille du code). Le client doit donc être prêt à récupérer plus que ce qu'il a demandé.

Si on prend cette ressource Core :

```
</sensors/temp>;rt="temperature-c";if="sensor",</sensors/light>;rt="light-lux";if="sensor"
```

Alors une requête HTTP `GET /.well-known/core?rt=light-lux` renverra (si le serveur gère les filtres) uniquement :

```
</sensors/light>;rt="light-lux";if="sensor"
```

Des questions de sécurité à garder en tête ? Non, pas trop, dit la section 6, qui rappelle qu'un serveur est autorisé à donner des informations différentes selon le client, et rappelle aussi que ne **pas** mettre un URI dans `/.well-known/core` n'apporte pas grand'chose, question sécurité (un attaquant peut le deviner).

Enfin, en section 7, les nombreux enregistrements à l'IANA nécessaires :

— `core` dans le registre des bien connus <`https://www.iana.org/assignments/well-known-uris/well-known-uris.xml`>,
— `hosts` (« héberge ») dans le registre des types de liens <`https://www.iana.org/assignments/link-relations/link-relations.xml`>,
— `application/link-format` dans la liste des types MIME <`https://www.iana.org/assignments/media-types/application/index.html`>,
— Et le tout nouveau registre des paramètres CoRE <`https://www.iana.org/assignments/core-parameters/core-parameters.xml`>, pour l'instant presque vide. Les nouvelles valeurs seront enregistrées en suivant, dans le RFC 5226 les politiques « "*IETF review*" » s'ils commençent par `core`, et, pour les autres, selon la politique « Spécification nécessaire ».

Les implémentations de ce RFC semblent surtout se trouver dans les bibliothèques COAP comme libcoap <`http://libcoap.sourceforge.net/`>, bien que COAP soit un protocole, pas un format. Voir le fichier `net.c` de cette bibliothèque. Une liste se trouve en bas de l'article Wikipédia.

TinyOS semble avoir un client capable de tester un `/.well-known/core`, voir <`http://docs.tinyos.net/tinywiki/index.php/CoAP`>. Par contre, je n'ai pas trouvé de validateur en ligne.