RFC 10008 : The HTTP QUERY Method

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 16 juin 2026

Date de publication du RFC : Juin 2026

https://www.bortzmeyer.org/10008.html

Ce n'est pas tous les jours qu'on normalise une nouvelle méthode HTTP. Bienvenue, donc, à QUERY, qui rejoint des méthodes bien plus anciennes comme GET et POST. QUERY peut être décrit comme « GET mais avec un corps dans la requête ». Comme GET, elle est idempotente et donc sûre à répéter.

L'idée est de pouvoir interroger, par exemple, un service de recherche (vous verrez un exemple plus loin sur mon blog). Sans QUERY, on envoyait quelque chose du genre :

```
GET /feed?q=foo&limit=10&sort=published HTTP/1.1
```

On est donc obligés de mettre les paramètres dans l'URL. Cela peut poser problème si les paramètres sont nombreux et de grande taille, cela oblige à les pourcent-encoder et cela peut poser des problèmes de vie privée (l'URL demandé a des chances d'être enregistré dans un journal).

Des gens utilisent donc POST pour une recherche, bien qu'il n'ait pas la bonne sémantique :

```
POST /feed HTTP/1.1

q=foo&limit=10&sort=published
```

Mais on ne voit plus que la requête est idempotente. Un navigateur Web n'osera pas la répéter ou bien demandera confirmation à l'utilisateur. Et on ne pourra pas facilement mémoriser le résultat (puisque le client ne sait pas si la requête n'a pas d'effets de bord). QUERY résout le problème :

```
QUERY /feed HTTP/1.1

q=foo&limit=10&sort=published
```

La méthode est idempotente, le résultat peut être mémorisé.

La section 2 du RFC décrit avec précision QUERY. À lire si vous écrivez des clients ou des serveurs qui l'utilisent. Par exemple, puisque QUERY, contrairement à GET, inclut un corps dans la requête, le client doit indiquer le type de média utilisé, et sans se tromper, sinon le serveur lui renverra un 400. (Et un 415 si le type est bien là mais que le serveur ne le connait pas.) Autre chose à noter : en cas de redirection, le client ne doit pas changer de méthode (alors qu'on pouvait changer un POST en GET si la redirection était faite avec 301 ou 302). Autrement, QUERY ressemble beaucoup dans son comportement à GET. QUERY est désormais enregistré dans le registre des méthodes HTTP <https://www.iana.org/assignments/http-methods/http-methods.xml#methods>.

Un serveur HTTP qui met en œuvre QUERY n'accepte pas forcément n'importe quel format en entrée. Pour documenter ce qu'il accepte comme corps de la requête, notre RFC introduit un nouveau champ HTTP <https://www.iana.org/assignments/http-fields/http-fields.xml#field-names>, `Accept-Query:` (section 3) qui est la liste des types de média acceptés.

Vous avez plein d'exemples de requêtes et de réponses dans l'annexe A.

Il y a une mise en œuvre de QUERY sur ce blog, pour fournir un moteur de recherche <https://www.bortzmeyer.org/moteur-recherche.html> des articles. L'URL est `https://www.bortzmeyer.org/m` et voici un exemple d'utilisation avec curl :

```
% curl --request QUERY --data query=framasoft https://www.bortzmeyer.org/methodquery

    Query of "framasoft" OK

    https://www.bortzmeyer.org/capitole-du-libre-2023.html "Capitole du Libre 2023, et mon exposé sur la cer
    ...
```

Vous pouvez avoir une documentation plus détaillée de ce service au début de son code source (en Python), . D'autre part, si vous n'aimez pas curl et que vous préférez un programme en Python, essayez ce client : . (Par contre, pas de formulaire Web pour utiliser ce service, car je ne connais pas de navigateur qui gère QUERY.)

En parlant de curl, notez que, lorsqu'il suit une redirection HTTP (option `--location`), il ne transmet pas actuellement <https://github.com/curl/curl/pull/16543> le corps de la requête, ce qui casse ce service.

La création de cette nouvelle méthode (ce qui est rare, je crois que la précédente avait été PATCH dans le RFC 5789[1] il y a quinze ans) a pris du temps. Le premier projet avait été rédigé en 2015 <https:

1. Pour voir le RFC de numéro NNN, `https://www.ietf.org/rfc/rfcNNN.txt`, par exemple `https://www.ietf.org/rfc/rfc5789.txt`

//datatracker.ietf.org/doc/draft-ietf-httpbis-safe-method-w-body/> et le travail a connu plusieurs interruptions. Une des discussions avait porté sur le nom de la méthode <https://github.com/httpwg/http-extensions/issues/1614>, qui aurait pu s'appeler SEARCH (réutilisant une méthode normalisée dans le RFC 5323). L'annexe B du RFC discute le choix qui a été fait.

Une autre discussion portait sur le code de retour HTTP, un problème classique de tous les services tournant sur HTTP : si la requête est bien transmise et traitée mais qu'on n'a pas de résultat, doit-on quand même renvoyer le 200, qui signifie que tout s'est bien passé ? Avec GET, on utilise souvent 404 dans ce cas, mais c'est parce que le terme de recherche est dans l'URL, ce qui n'est plus le cas ici. On aurait pu aussi avoir un nouveau code commençant par 2. Finalement, le choix a été de renvoyer 200 quand la requête est bien arrivée et que le moteur de recherche a fonctionné, même s'il n'a rien trouvé. (Une discussion analogue avait eu lieu pendant le développement de DoH. Le RFC 8484 avait finalement décidé de répondre 200 même si le nom de domaine demandé n'existait pas.)

Ah, et si vous voulez superviser votre service HTTP utilisant QUERY, le programme check_http <https://www.monitoring-plugins.org/doc/man/check_http.html> des "*monitoring plugins*" <https://www.monitoring-plugins.org/> le permet. Voici un exemple de configuration pour Icinga :

```
vars.http_vhosts["query"] = {
    http_uri = "/methodquery"
    http_vhost = "www.bortzmeyer.org"
    http_ssl = true
    http_sni = true
    http_method = "QUERY"
    # Notez que le nom de la variable n'est pas très heureux.
    http_post = "query=foobar"
    http_content_type = "application/x-www-form-urlencoded"
    http_string = "foobar\" OK"
    http_timeout = 15
}
```