

Le Protocole HTTP

par [Mathieu Lemoine](#)

Date de publication : 25/06/2006

Dernière mise à jour : 09/10/2006

Protocole parmi les plus courants, le protocole HTTP est très souvent méconnu pour ne pas dire inconnu des webmasters et des développeurs d'applications Web en général. Ce tutoriel vous aidera à acquérir les connaissances de base (ainsi que certaines autres plus avancées) concernant ce protocole.

Notes

Remerciements

Introduction

I - Généralités

I-A - Présentation

I-B - Structure

I-C - Caractères Interdits ou Réservés

I-D - L'URL-Encoding

II - Les Requêtes HTTP

II-A - La ligne d'Introduction

La Méthode

La Page

La Version

Les Méthodes

II-B - Les en-têtes de requête

II-C - Le corps de la requête

III - Les Réponses HTTP

III-A - La ligne d'Introduction

Informations : les 1xx

Succès : les 2xx

Redirection : les 3xx

Requête Invalide : les 4xx

Erreur Serveur : les 5xx

D'autres codes status

III-B - Les en-têtes de réponse

IV - Divers

IV-A - Le cache : qu'est-ce que c'est ?

IV-B - Et PHP/ASP/JSP/les CGI dans tout ça ?

IV-C - Et (D)(X)HTML/JavaScript/ActiveX dans tout ça ?

IV-D - Une connexion = Un échange ?

IV-E - Les cookies : le clignotant du Web ?

IV-F - Voir le protocole HTTP

IV-G - Voir aussi

Notes

Remerciements

Merci à [Yogui](#) et à [neguib](#) pour leurs relectures et leurs conseils.

Introduction

Cet article présente le protocole HTTP (HyperText Transfert Protocole) d'une manière simple et, normalement, compréhensible pour n'importe qui ayant un minimum de connaissances en matière de Théorie des Réseaux Informatiques (notamment la structure client/serveur, échanges binaires/échanges textes, ...).

Sur l'Internet, de nombreux protocoles sont utilisés. Le protocole HTTP est l'un des plus courants. C'est notamment celui qui est utilisé pour la navigation sur les sites Web. Par exemple, si vous lisez actuellement la version en ligne de cet article, votre navigateur a eu recours au protocole HTTP pour récupérer la page Web, les différentes images, feuilles de styles, etc. Ce protocole est relativement simple dans sa forme et ses utilisations fondamentales devraient être connues de tous les développeurs Web.

I - Généralités

I-A - Présentation

Il s'agit d'un protocole reposant sur un unique échange initié par le client. Il est synchrone : le serveur ne peut commencer à répondre avant que le client ait fini d'envoyer la requête.

Il en existe trois versions : 0.9 (1991), 1.0 (février 1997) et 1.1 (octobre 2000). La version 0.9 est complètement obsolète et nous ne l'étudieront pas ici. De nos jours, la version 1.0 est très rarement utilisée ; elle est d'ailleurs obsolète. Les principaux changements entre les versions 1.0 et 1.1 sont l'ajout de 2 types de requêtes ainsi que la possibilité d'héberger plusieurs sites Web sur un même serveur dans la version 1.1.

L'échange entre le client et le serveur se fait en mode texte. Le charset généralement utilisé est l'US-ASCII sur 8 bits. Il est cependant possible que cet encoding soit modifié selon le client ou le serveur.

I-B - Structure

Les requêtes et les réponses sont bâties sur le même modèle :

```
{Ligne d'introduction}{SEP}
{En-têtes séparées par des {SEP} }
{SEP}{SEP}
{Corps}
```

En théorie, le seul élément capable de différencier une requête d'une réponse, c'est la *Ligne d'introduction*.

En fait, certaines en-têtes sont plutôt caractéristiques des requêtes, tandis que d'autres sont plutôt utilisées pour les réponses. Les requêtes ont cette forme :

```
{Nom}{HSEP}{Valeur}
```

Le séparateur *HSEP* représente la combinaison deux points + espace " : ". Le protocole HTTP définit un ensemble d'en-têtes standard. Des en-têtes supplémentaires peuvent être ajoutées, à la condition que leur nom commence par "X-".

I-C - Caractères Interdits ou Réservés

Certains caractères sont interdits ou réservés à certains endroits du message :

nom du caractère	code US-ASCII	notation courante	réserves et interdictions
retour chariot (Carriage Return, CR)	0x0D (13)	\r	Il fait partie du séparateur <i>SEP</i> . Son utilisation à l'intérieur de la <i>Ligne d'Introduction</i> et des <i>en-têtes</i> est donc <u>interdite</u> .
nouvelle ligne (New Line ou Line Feed,	0x0A (10)	\n	Il fait partie du séparateur <i>SEP</i> .

nom du caractère	code US-ASCII	notation courante	réserves et interdictions
LF)			Son utilisation à l'intérieur de la <i>Ligne d'Introduction</i> et des <i>en-têtes</i> est donc <u>interdite</u> .
point d'interrogation	0x3F (63)	?	Il est utilisé à certains endroits de la <i>Ligne d'Introduction</i> . Son utilisation à l'intérieur de celle-ci est donc <u>réglementée</u> .
égal	0x3D (31)	=	Il est utilisé à certains endroits de la <i>Ligne d'Introduction</i> . Son utilisation à l'intérieur de celle-ci est donc <u>réglementée</u> .
esperluette (et commercial)	0x26 (38)	&	Il est utilisé à certains endroits de la <i>Ligne d'Introduction</i> . Son utilisation à l'intérieur de celle-ci est donc <u>réglementée</u> .
dièse	0x23 (35)	#	Il est utilisé à certains endroits de la <i>Ligne d'Introduction</i> . Son utilisation à l'intérieur de celle-ci est donc <u>réglementée</u> .
pourcentage	0x25 (37)	%	Il est utilisé dans le cadre de l' <i>URL-Encoding</i> . Son utilisation à l'intérieur de celui-ci est donc <u>réglementée</u> .
deux points	0x3A (58)	:	Il sont utilisés dans les <i>en-têtes</i> . Son utilisation à l'intérieur de celles-ci est donc <u>réglementée</u> .
espace (Space, SP)	0x20 (32)	' ' ou " "	Il est utilisé dans la <i>Ligne d'Introduction</i> et les <i>en-têtes</i> . Son utilisation à l'intérieur de celles-ci est donc <u>réglementée</u> .

Le séparateur SEP correspond en fait au duo "\r\n" (Carriage Return, New Line : CRLF)

I-D - L'URL-Encoding

L'URL-Encoding permet d'insérer les caractères réservés, ou des caractères accentués, dans certaines parties des messages HTTP.

Cet encodage consiste simplement à remplacer un caractère interdit par : le caractère pourcentage suivi des deux caractères représentant le code hexadécimal US-ASCII du caractère à remplacer.

Exemple : le séparateur *SEP* (\r\n) deviendrait : %0D%0A

Pour décoder une chaîne URL-Encodée, il suffit de remplacer le triplet %?? par le caractère dont le code US-ASCII correspond.

II - Les Requêtes HTTP

Dès que le client est connecté au serveur, il envoie sa requête. C'est en ce sens que l'échange est initié par le client.

Nous allons voir les spécificités des requêtes HTTP.

II-A - La ligne d'Introduction

Comme précisé auparavant, le seul point qui change catégoriquement entre requête et réponse HTTP, c'est cette Ligne d'Introduction. Voici la forme qu'elle a pour une requête :

```
{Méthode}{SP}{Page}{SP}{Version}
```

Le séparateur *SP* correspond à l'espace.

La Méthode

- GET
- HEAD
- POST
- PUT
- DELETE
- OPTIONS
- TRACE
- CONNECT

On parle aussi parfois de type de requêtes.

Les Méthodes autres que **GET** et **POST** sont propres à la version 1.1 de HTTP.

Ces méthodes seront détaillées un peu plus loin.

La Page

Le spécificateur *Page* désigne le chemin vers la ressource chargée de traiter, ou de correspondre à la requête à partir de la racine du site.

Il doit commencer par un slash (/) et la chaîne doit être URL-Encodée.

- Les paramètres d'URL (paramètres GET)
- L'identificateur de fragment (*fragment identifier*)

Les paramètres d'URL sont indiqués avec cette syntaxe :

```
{Nom}{EQ}{Valeur}
```

Le *Nom* et la *Valeur* sont URL-Encodés. Forcément, il n'y a pas d'espace. Le séparateur *EQ* correspond au caractère égal.

Chaque paramètre est séparé du suivant par un esperluette.

La chaîne ainsi formée est placée à la suite du chemin, le tout étant séparé par un point d'interrogation.

Le *fragment identifiant* permet de désigner une partie précise du document. Il n'a pour le moment que peu (voire pas) de signification pour le serveur, car il est surtout utile au client. Par exemple, c'est lui qui est utilisé pour désigner une ancre dans un document HTML.

Le langage *XPointer* est prévu pour être notamment utilisé dans le *fragment identifiant*.

Il doit également être URL-Encodé. Il est ajouté à la suite des paramètres d'URL (ou du chemin s'il n'y a pas de paramètre d'URL) et il en est séparé par un dièse.

La Version

- HTTP/1.0 pour la version 1.0
- HTTP/1.1 pour la version 1.1

Les Méthodes

Nous allons maintenant voir un peu plus en détail les 4 méthodes proposées par la version 1.1 du protocole HTTP. Les informations sur les méthodes GET et POST sont également valables pour la version 1.0.

- **GET**

Cette méthode est la plus courante, il s'agit normalement d'une simple requête de téléchargement d'un document. Deux requêtes GET portant sur le même document devraient retourner des réponses sémantiquement identiques (certaines en-têtes pouvant influencer sur le comportement du serveur, les réponses peuvent ne pas être totalement identiques).

Aucune donnée à traiter ne peut être envoyée au serveur par cette méthode. Il est par contre possible d'ajouter les paramètres d'URL (aussi nommés paramètres GET). Le corps de la requête DOIT être vide, et le document spécifié dans la requête (la Page) est celui qui doit être retourné.

- **HEAD**

Cette méthode est similaire et presque équivalente à la méthode GET. C'est en fait la réponse finale du serveur qui est tronquée.

La réponse à une requête HEAD est la réponse à la requête GET qui lui est similaire, sauf que le corps de la réponse HTTP n'est pas transmis. Cela permet souvent d'économiser beaucoup de bande passante.

- **POST**

La Méthode POST est la méthode de base pour demander un traitement d'informations au serveur. Ces requêtes sont censées mettre en jeu des mécanismes propres au serveur et provoquant des communications avec d'autres modules, voire d'autres serveurs, pour effectuer le traitement des dites données. De ce fait, il est tout à fait probable que deux requêtes POST identiques reçoivent des réponses différentes ou même sémantiquement opposées.

Les données à traiter sont spécifiées dans le corps de la requête.

Le document désigné par la requête via la page est la ressource qui doit traiter les données et générer la réponse.

- **PUT et DELETE**

Ces méthodes sont censées permettre l'upload (le chargement sur le serveur) ou la suppression d'un document sans passer par un serveur FTP ou autre. Bien évidemment, cela peut provoquer des remplacements de fichiers, et donc de très grosses failles de sécurité sur un serveur. De ce fait, la plupart des serveurs Webs requièrent une configuration spéciale indiquant une ressource ou un document chargé de traiter ces requêtes.

Le document désigné par la requête est celui qui doit être remplacé (ou créé), et le contenu du document est dans le corps de la requête.

En théorie, les paramètres d'URL et le *fragment identifier* devraient être interdits ou ignorés par le serveur. En pratique, ils sont généralement transmis à la ressource chargée de traiter la requête.

- **OPTIONS et TRACE**

Ces méthodes permettent au client de demander certaines informations sur le serveur.

Tous les serveurs ne les implémentent pas forcément.

- **CONNECT**

Cette méthode est censée être utilisée pour demander une utilisation du serveur en tant que proxy.

Tous les serveurs ne les implémentent pas forcément.

II-B - Les en-têtes de requête

Nous allons ici présenter certaines en-têtes utilisées dans les requêtes HTTP. Elles ne sont pas forcément spécifiques aux requêtes, mais ont une signification particulière dans le cadre d'une requête.

- **Host**

Cette en-tête est la seule obligatoire pour les requêtes HTTP/1.1, c'est elle qui permet d'héberger plusieurs sites Web sur un même serveur.

Sa valeur est le domaine du site Web. Elle est généralement spécifiée juste après la Ligne d'Introduction.

- **User-Agent**

Cette en-tête permet d'indiquer la signature du programme effectuant la requête. C'est une chaîne de caractères qui permet d'identifier le programme. En général, il s'agit du nom complet du programme et de sa version.

- **Content-type et Content-length**

Ces deux en-têtes ne peuvent être spécifiées que dans le cadre d'une requête POST ou PUT. Elles indiquent respectivement le type MIME et la taille en octets du corps de la requête. Si elles ne sont pas spécifiées, c'est le serveur qui est seul responsable de leur éventuelle valeur par défaut.

- **Cookie**

Cette en-tête permet au client de fournir un cookie au serveur.

Sa valeur est simplement le nom et la valeur du cookie, séparés par un égal.

- **D'autres en-têtes**

De nombreuses autres en-têtes ont été prévues dans le protocole HTTP. Elles servent par exemple à préciser la gestion du cache, la langue ou le format préféré pour la réponse, l'authentification du client, etc. Les Requests For Comments (RFC) du protocole HTTP vous fourniront toutes les en-têtes [pour ses versions 1.0 et 1.1]. Toute en-tête non reconnue par le serveur doit théoriquement être ignorée.

II-C - Le corps de la requête

Le corps de la requête doit être vide pour les requêtes GET, HEAD, DELETE, CONNECT, TRACE et OPTIONS (dans le dernier cas, il est laissé éventuellement rempli pour de futures versions du protocole HTTP).

Dans le cas des requêtes POST et PUT, il correspond aux données à traiter. Il n'y a pas de caractères réservés ou interdits dans le corps de la requête au niveau du protocole HTTP. Cependant, les données doivent être conformes au format attendu par la ressource responsable de leur traitement.

III - Les Réponses HTTP

Quand le serveur a fini de recevoir la requête, il la traite et renvoie la réponse appropriée (éventuellement une erreur).

Les réponses sont bâties sur un modèle similaire à celui des requêtes. Voyons leurs spécificités.

III-A - La ligne d'Introduction

Les lignes d'Introduction des réponses HTTP ont une structure plus simple que celle des requêtes, mais tout aussi riches en informations. Le format est :

```
{Version}{SP}{Code Status}{SP}{Phrase Status}
```

La *Version* est la même que pour la requête.

Le *Code Status* est un code sur trois chiffres qui indique le status de la requête. Nous y reviendrons plus tard.

Le séparateur *SP* est un espace.

La *Phrase Status* est simplement une phrase permettant de rendre le status plus lisible pour un humain. Elle est purement indicative et n'a aucune valeur informative sur le plan du protocole, elle est d'ailleurs facultative. Nous indiquerons cependant les *Phrases Status* (Reason Phrase) proposées par les RFC pour les codes que nous détaillerons.

Les Codes Status sont regroupés en familles, qui sont au nombre de cinq pour les versions 1.0 et 1.1 de HTTP. La famille est indiquée par le premier chiffre du code (celui des centaines), il va de 1 à 5.

Tout code inconnu doit être traité par le client comme le code de base de la famille (X00) et être présenté à l'utilisateur.

Si la famille est inconnue, le code doit être traité comme un code Erreur Serveur (famille 5xx) et être indiqué à l'utilisateur.

Informations : les 1xx

Ces codes sont là simplement pour permettre au serveur d'envoyer une notification.

- **100 Continue**

Ce code status est rarement utilisé et informe simplement que la partie de la requête qui a déjà été reçue est valide. Il n'est pas envoyé par défaut, mais seulement dans des cas précis.

- **101 Switching protocol**

Ce code status permet de changer le protocole ou la version du protocole utilisé lors de la communication. Le nouveau protocole à utiliser est spécifié par l'en-tête Upgrade.

Succès : les 2xx

Ces codes indiquent que tout s'est bien déroulé et que la ressource demandée est renvoyée.

- **200 OK**

Tout est bon, c'est la réponse la plus souvent employée.

- **201 Created**

Peut être utilisé par exemple dans le cadre d'une requête PUT pour indiquer que le document a bien été uploadé.

- **204 No Content**

La requête s'est bien déroulée, mais le corps de la réponse est vide.

- **206 Partial Content**

Ce code est généralement utilisé dans le cadre d'une récupération de téléchargement, ou de l'utilisation du cache. Seule une partie du document demandé est renvoyée.

Redirection : les 3xx

Ces codes indiquent que la ressource demandée n'est plus à l'emplacement indiqué. Ils sont très utilisés pour les redirections. L'en-tête, accompagnant la redirection et qui indique le nouvel emplacement de la ressource, est alors l'en-tête Location.

- **300 Multiple Choice**

Ce code est utilisé quand on peut trouver plusieurs versions de la ressource (différence de format ou de langue par exemple).

- **301 Moved Permanently**

Quand une ressource est déplacée définitivement, c'est ce code qui permet d'indiquer le déplacement notamment aux moteurs de recherche. La requête ayant généré l'erreur doit alors être renvoyée pour correspondre avec la nouvelle ressource.

- **307 Temporary Redirect**

Ce code permet d'indiquer une redirection temporaire.

Requête Invalide : les 4xx

Ces codes indiquent une erreur de la part du client, ressource invalide, authentification invalide, etc. La requête doit alors être modifiée et éventuellement retransmise pour pouvoir être traitée.

- **400 Bad Request**

Erreur générique.

- **401 Unauthorized ou Authorization required**

Le client n'est pas censé avoir accès à la requête au vu de son niveau actuel d'identification. Il doit s'identifier de manière correcte. S'il ne peut remplir les conditions d'identification, alors les requêtes suivantes aboutiront à une erreur 403.

- **403 Forbidden**

La ressource est interdite au client.

- **404 Not Found**

La fameuse erreur 404, elle indique que la ressource demandée n'a pu être trouvée.

- **405 Method Not Allowed**

Le client n'est pas censé pouvoir envoyer ce type de requête, une authentification est certainement nécessaire.

Erreur Serveur : les 5xx

Ces codes sont utilisés en cas d'erreur de la part du serveur, en cas de surcharge, d'erreur de configuration, etc. Le plus simple est d'attendre un peu et de retenter plus tard : si l'erreur persiste, contactez l'administrateur du serveur.

- **500 Internal Server Error**

Erreur Interne au serveur, il n'est pas en état de répondre actuellement.

- **501 Not Implemented**

Certaines fonctionnalités requises par les en-têtes ou la méthode employées ne sont pas supportées par le serveur.

- **503 Service Unavailable**

Utilisé par exemple quand le serveur est surchargé.

- **505 HTTP Version Not Supported**

Le serveur ne supporte pas la version du protocole HTTP qui a été utilisée.

D'autres codes status

Tous les codes n'ont bien entendu pas été listés ici, certains ont même été invalidés lors du passage de HTTP/1.0 à HTTP/1.1 (306 par exemple), ou bien sont réservés pour des versions ultérieures du protocole (402 par exemple).

III-B - Les en-têtes de réponse

Voici les principales en-têtes utilisées lors des réponses.

- **Content-type et Content-length**

Ces deux en-têtes sont censées indiquer le type MIME et la taille en octets du corps de la réponse. De plus, l'en-tête Content-type peut indiquer le charset utilisé dans le corps de la réponse (dans le cadre d'un type texte). Il est alors indiqué par la mention "charset=" suivie du nom du charset. Il suit le type MIME, en est séparé par un point-virgule. Si elles ne sont pas indiquées, c'est le client qui est seul responsable de leur éventuelle valeur par défaut.

- **Location**

Cette en-tête permet d'indiquer une redirection. A sa réception, le client est généralement censé renvoyer une requête sur l'adresse indiquée. Ce comportement dépend du code status renvoyé avec la réponse.

- **Set-Cookie**

Cette en-tête permet d'indiquer au client des cookies à stocker. Sa valeur peut prendre une forme assez complexe. La forme par défaut est celle de l'en-tête de requête *Cookie*. Les formes plus évoluées consistent à l'indication d'informations telles que : la date de péremption (Expires), le domaine d'application (Domain), le chemin d'application (Path), etc. Toutes ces informations sont indiquées à la suite du couple nom/valeur de la même manière (nom de l'information, égal, valeur) et séparés entre elles et de celui-ci par un point-virgule.

- **D'autres en-têtes**

Bien sûr, toutes les en-têtes ne sont pas listées ici. Vous trouverez sûrement une description exhaustive dans la RFC.

IV - Divers

IV-A - Le cache : qu'est-ce que c'est ?

Afin de réduire la bande passante utilisée, le protocole HTTP met à disposition des clients le support d'un mécanisme de sauvegarde des données. Ainsi, sous certaines conditions, les clients sont autorisés à stocker les réponses du serveur afin de ne pas devoir redemander toute la page au serveur. Tout un ensemble d'en-têtes tant pour les requêtes que pour les réponses est dédié à cette gestion. De plus, le rôle de la méthode HEAD entre exclusivement dans ce cadre.

IV-B - Et PHP/ASP/JSP/les CGI dans tout ça ?

PHP, ASP, JSP et les CGI (aussi bien les scripts CGI et les CGI-bin) sont des langages ou des applications côté serveur, c'est le serveur et LUI SEUL qui se doit de les gérer. Il peut leur permettre d'interférer avec la réponse HTTP envoyée, via les headers ou le corps par exemple, mais cela relève de sa seule responsabilité et doit être transparent pour le client.

IV-C - Et (D)(X)HTML/JavaScript/ActiveX dans tout ça ?

(D)(X)HTML, JavaScript, ActiveX, CSS, etc. sont des langages ou des applications côté client, ils sont contenus dans le corps de la réponse HTTP et doivent être transparents pour le serveur. Ils ne peuvent normalement pas interférer avec le protocole HTTP puisqu'ils entrent en jeu après réception de la réponse par le client. Cependant, ils peuvent parfois interférer sur la création de la requête (formulaires (X)HTML) ou sur l'utilisation de l'échange HTTP (AJAX).

IV-D - Une connexion = Un échange ?

En théorie : OUI!

Ensuite, HTTP propose des mécanismes d'optimisation qui permettent de conserver la connexion entre le serveur et le client. Ainsi, nous pouvons charger des ressources annexes ou faire d'autres requêtes HTTP, car la mise en place d'une connexion est relativement coûteuse (surtout en temps d'exécution). Dans ce cas encore, ce sont des en-têtes spécifiques qui sont chargées de la gestion de cette connexion.

IV-E - Les cookies : le clignotant du Web ?

La gestion des cookies est assez complexe :

Côté client, on peut, en théorie, y accéder en permanence puisque c'est là qu'ils sont stockés.

Côté serveur, le comportement doit être étudié au cas par cas : configuration du client, configuration du serveur, langage utilisé côté serveur, configuration du module coté serveur sont autant de paramètres qui peuvent influencer sur leurs réactions.

IV-F - Voir le protocole HTTP

Si vous souhaitez tester le protocole HTTP par vous-même, vous pouvez utiliser *telnet*. Ce programme est disponible sur la plupart des plate-formes. Pour cela faites :

```
telnet {nom de domaine} 80
```

Puis tapez votre requête. En théorie, vous devriez voir votre requête arriver.

IV-G - Voir aussi

- [RFC HTTP/1.0 \(Version française\)](#)
- [RFC HTTP/1.1](#)