Network Working Group
Request for Comments: 1945
Category: Informational

T. Berners-Lee, MIT/LCS
R. Fielding, UC Irvine
H. Frystyk, MIT/LCS

May 1996

French translation by V.G. FREMAUX / Ecole Internationale des Sciences du Traitement de l'Information
(International Graduate School of Computer Sciences)

HyperText Transfer Protocol HTTP/1.0

INTERNET
STANDARDS & OFFICIAL DOCUMENTS

Standard	✗
Informational	✓
Proposal	✗

Note du Traducteur :

Le texte suivant est la traduction intégrale de la spécification HTTP 1.0, telle qu'éditée par les auteurs originaux du protocole, sans ajouts, commentaires, ni omissions. Ce document a valeur informative, selon la procédure courante d'enregistrement des documents au sein du W3C. Il n'a de ce fait pas valeur de "norme", et, s'il peut être considéré comme document de référence, il ne peut par contre octroyer une légitimité quelconque à un quelconque développement, ni servir de preuve ni de garant d'une conformité quelle qu'elle soit.

Statut de ce mémo

Ce mémo est une information à destination de la communauté Internet. Il ne peut être considéré comme une spécification officielle d'aucun protocole Internet. Sa distribution n'est pas limitée.

Note IESG:

L'IESG espère que ce document sera rapidement remplacé par une norme officielle.

Contexte

L'Hypertext Transfer Protocol (HTTP) est un protocole de niveau application suffisamment léger et rapide, pour la transmission de documents distribués et multimédia à travers un système d'information multi-utilisateurs. Il s'agit d'un protocole générique, orienté objet, pouvant être utilisé pour de nombreuses tâches, dans les serveurs de nom et la gestion d'objets distribués par l'extension de ses méthodes de requête (commandes). Une caractéristique d'HTTP est son typage des représentations de données, permettant la mise en oeuvre de systèmes indépendants des données y étant transférées.

HTTP a été utilisé à l'initiative du World-Wide Web depuis 1990. Cette spécification décrit le protocole de base référencé sous l'appellation "HTTP/1.0".

Table des Matières

1. Introduction

1.1 Objectif

L'Hypertext Transfer Protocol (HTTP) est un protocole de niveau application suffisamment léger et rapide, pour la transmission de documents distribués et multimédia à travers un système d'information multi-utilisateurs. HTTP à été utilisé à l'initiative du Word-Wide Web dès 1990. Cette spécification décrit les fonctionnalités le plus souvent rencontrées dans les implémentation "client/serveur HTTP/1.0". Elle est divisée en deux section. Les fonctions usuellement exploitées de HTTP sont décrites dans le corps de ce document. Les autres fonctions, moins utilisées sont listées dans l'annexe D.

Les systèmes d'information pratiques nécessitent des fonctions plus évoluées que la simple récupération de données, parmi lesquelles on trouve la possibilité d'effectuer des recherches, les fonctions de remise à jour, et d'annotation. HTTP permet d'utiliser un ensemble non exhaustif de méthodes pour définir l'objet d'une requête. Il s'appuie essentiellement sur la normalisation des Uniform Resource Identifier (URI) [2], soit sous forme de "site" (URL) [4] ou de noms (URN) [16], pour indiquer l'objet sur lequel porte la méthode. Les messages sont transmis sous une forme similaire à celle de la messagerie électronique (Mail) [7] et des extensions Multipurpose Internet Mail Extensions (MIME) [5]. HTTP est aussi un protocole de communication générique entre des "utilisateurs" et des routeurs/proxies vers d'autres protocoles, tels que SMTP [12],

NNTP [11], FTP [14], Gopher [1], et WAIS [8], permettant une accès de base à des ressources hypermédia, et simplifiant l'implémentation d'interfaces utilisateur.

1.2 Terminologie

Cette spécification utilise un certain nombre de termes pour désigner les participants et les objets d'une communication HTTP.

Connexion

Un circuit virtuel s'appuyant sur une couche de transport pour la communication d'information entre deux applications.

Message

L'unité de base d'une communication HTTP, consistant en une séquence structurée d'octets définie à la Section 4 et transmis via la connexion.

Requête

Un message de requête HTTP (défini en Section 5).

Réponse

Un message de réponse HTTP (défini en Section 6).

Ressource

Un service ou objet du réseau pouvant être identifié par une URI (Section 3.2).

Entité

Une représentation particulière de données, ou la réponse d'une ressource de type service, incluse dans une requête ou une réponse. Une entité représente une métainformation sous la forme d'une en-tête et d'un contenu sous la forme d'un corps, ou "body".

Client

Un programme applicatif dont la fonction principale est d'émettre des requêtes.

Utilisateur

Le client qui a émis la requête. Celui-ci peut être un navigateur, un éditeur, un "spiders" (robot d'exploration), ou autre utilitaire.

Serveur

Un programme applicatif acceptant des connexions dans le but traiter des requêtes en délivrant une réponse.

Serveur origine

Le serveur dans laquelle se situe physiquement une ressource.

Proxy

Un programme intermédiaire qui cumule les fonctions de serveur et de client. Les requêtes sont soit traitées en interne ou répercutées, éventuellement converties, sur d'autres serveurs. Un proxy doit interpréter, puis reconstituer un message avant de le réemettre. Les proxies sont souvent utilisés comme portes d'accès côté utilisateur à des réseaux protégés par un "firewall" ou comme intermédiaire pour convertir des protocoles non supportés par l'utilisateur.

Gateway ou routeur

Un serveur jouant le rôle d'intermédiaire pour d'autres serveurs. Contrairement à un Proxy, un routeur reçoit les requêtes comme s'il était le serveur origine pour la ressource; le client n'étant pas nécessairement conscient de "communiquer" avec un routeur. Les routeurs sont souvent utilisés comme porte d'accès côté serveur d'un réseau protégé par "firewall" et comme convertisseur de protocole pour accéder à es ressources hébergées sur des systèmes non-HTTP.

Tunnel

Un tunnel est un programme applicatif servant de relais "transparent" entre deux connexions. Une fois actif, cet élément n'est plus considéré comme faisant partie de la connexion HTTP, bien qu'il soi issu d'une requête HTTP. Le tunnel cesse d'exister lorsque les deux sens de la connexion ont été fermés. Les tunnels sont utilisés lorsqu'une "porte" est nécessaire et que l'intermédiaire ne peut, ou ne doit pouvoir interpréter la communication.

Cache

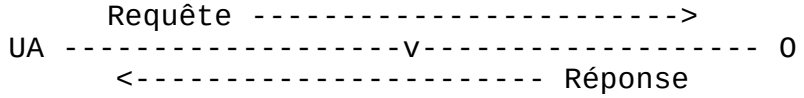
Un espace de stockage local destiné à enregistrer les réponses et le sous-système contrôlant ces enregistrements, leur relecture et leur effacement. Un cache enregistre des réponses dans le but de diminuer le temps d'accès et la charge du réseau lors de requêtes identiques ultérieures. Tout client ou serveur peut implémenter un cache, bien que le serveur ne puisse exploiter ce cache, faisant office de tunnel.

Un programme pouvant aussi bien être serveur que client; l'utilisation que nous ferons de ces termes s'appliquera à la situation qu'à le programme vis à vis d'une connexion particulière, plutôt qu'à ses possibilités effectives. De même tout serveur peut être à la fois serveur origine, proxy, routeur, ou tunnel, changeant alors de comportement en fonction des requêtes reçues.

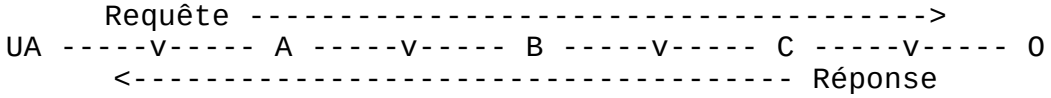
1.3 Fonctionnement global

Le protocole HTTP est basé sur un paradigme requête/réponse. Un client établit une connexion vers un serveur et lui envoie une requête sous la forme d'une méthode, d'une URI, du numéro de version, suivi d'un message de type MIME contenant les modificateurs de la requête, les informations sur le client, et éventuellement un corps. Le serveur répond par une ligne d'état, incluant la version de protocole et un message de succès ou d'erreur, suivi d'un message de type MIME contenant l'information sur le serveur, metainformation, et le corps éventuel.

La plupart des communications HTTP sont initiées par un utilisateur et consistent en une requête devant être appliquée (il s'agit d'une méthode) à une ressource sur son serveur origine. Dans le cas le plus simple, ceci peut être réalisé par une connexion simple (v) entre l'utilisateur (UA) et le serveur origine (O).



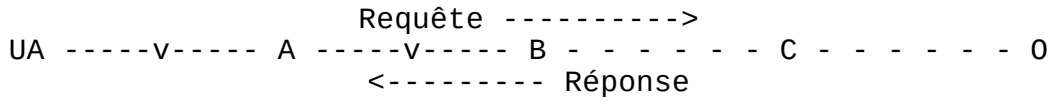
Une situation plus complexe peut apparaître lorsque un ou plusieurs intermédiaires sont présents dans la chaîne de communication. On trouvera trois types d'intermédiaires: les proxy, les routeurs, et les tunnels. Un proxy est un agent actif, recevant les requêtes destinées à une URI dans sa forme absolue, recomposant tout ou partie du message, et réémettant la requête transformée à destination de l'URI. Un serveur est un agent actif, agissant en tant que "surcouche" par rapport à d'autres serveurs et si nécessaire, traduisant le protocole à destination et en provenance du serveur visé. Un tunnel est un agent passif ne servant que de relais entre deux parties d'une même connexion, et de ce fait sans modifier le message; les tunnels sont utilisés lorsque le message doit traverser un intermédiaire (comme un "firewall") même si celui-ci ne peut interpréter le contenu des messages.



La figure ci-dessus montre le cas de trois intermédiaires (A, B, et C) entre l'utilisateur et le serveur origine. Une requête ou réponse passant d'un bout à l'autre doit transiter par quatre connexions. Cette distinction est important car certaines options d'une communication HTTP peuvent ne s'appliquer qu'à la connexion la plus proche, sans

même passage par un tunnel, et accessible uniquement des extrémités, ou au contraire à toutes les connexions impliquées dans la chaîne. Bien que ce schéma soit linéaire, chaque participant peut être engagé dans plusieurs communications simultanées. Par exemple, B peut être en train de recevoir de nombreuses requêtes de clients autres que A, et émettre des requêtes vers d'autres serveurs que C, tout en interprétant les requêtes issues de A.

Toute partie de communication n'étant pas un tunnel doit posséder un cache pour le stockage des requêtes. L'intérêt de cette pratique est que la chaîne requête/réponse peut être considérablement raccourcie si l'un des intermédiaires dispose déjà d'une instance de la ressource dans son cache. Le diagramme suivant montre le cas où B dispose dans son cache d'une copie d'une réponse précédemment envoyée par O (via C) et répond à une requête identique de UA (à noter que dans cet exemple, ni UA ni A ne disposent de cette même réponse en cache).



Toutes les réponses ne peuvent être "cachées", et certaines requêtes peuvent utiliser des modificateurs induisant un comportement particulier du cache. Certaines applications HTTP/1.0 utilisent des heuristiques pour décrire ce qui est ou non "cachable", mais ces règles ne sont pas standardisées.

Sur Internet, les communications HTTP s'appuient principalement sur le protocole de connexion TCP/IP. Le port utilisé par défaut est le port TCP 80 [15], d'autres ports pouvant être utilisés. Ceci n'exclut pas que HTTP puisse être implémenté au dessus d'un autre protocole d'Internet, ou sur d'autres types de réseau. HTTP nécessite seulement un transport fiabilisé; tout protocole garantissant la sécurité de transmission peut être utilisé pour supporter HTTP, la place qu'occupent les données des requêtes et réponses HTTP/1.0 dans les unités de transmission de ces protocoles restant en dehors du contexte de ce document.

Excepté pour des applications expérimentales, la pratique courante spécifie qu'une connexion doit être initiée par un client avant transmission de la requête, et refermée par le serveur après délivrance de la réponse. Les deux côtés, client et serveur, doivent être préparés à ce que la connexion soit coupée prématurément, suite à une action de l'utilisateur, une temporisation automatique, ou une faute logicielle, et doivent apporter une réponse prévisible à cette situation. Dans tous les cas, la fermeture d'une connexion qu'elle qu'en soit la raison est assimilable à la conclusion de la requête, quel que soit l'état.

1.4 HTTP et MIME

HTTP/1.0 exploite un grand nombre d'implémentation prévues pour les MIME, tels que définis dans la RFC 1521 [5]. L'appendice C décrit comment HTTP l'utilisation des "Internet Media Types" typiquement utilisés par la messagerie électronique, et indique les différences de comportement.

2. Conventions de notation et Grammaire générique

2.1 BNF étendue

Tous les mécanismes évoqués sont décrits en prose et sous forme Backus-Naur étendue (BNF) similaire à celle utilisée dans la RFC 822 [7]. Les développeurs devront être familiarisés avec cette notation afin de comprendre cette spécification. La notation Backus-Naur comprend les allégations suivantes :

nom = définition

Le nom d'une règle est le nom lui-même (sans "<" ni ">" englobants) et est séparé de sa définition par le symbole "=". Le caractère espace n'a de signification que lorsqu'un retrait indique qu'une définition s'étend sur plusieurs lignes. Certaines règles de base sont en majuscules, comme SP, LWS, HT, CRLF, DIGIT, ALPHA, etc. Les ouvertures et fermetures "<" et ">" ne sont utilisées que lorsqu'une discrimination des règles est indispensable à l'intérieur d'une définition.

"littéral"

Un texte littéral est entre doubles guillemets. Sauf mention contraire, la casse du texte n'est pas considérée.

règle1 | règle2

Les éléments séparés par une barre ("I") constituent une alternative, ex., "oui | non" acceptera "oui" ou "non".

(règle1 règle2)

Les éléments à l'intérieur de parenthèses sont considérés comme un seul élément. Ainsi, "(elem (foo | bar) elem)" permet les séquence "elem foo elem" et "elem bar elem".

***règle**

Le caractère "*" précédent un élément indique sa répétition. La forme complète de cette notation est "<n>*<m>element" indiquant au moins <n> et au plus <m> occurrences de "element". Les valeurs par défaut sont 0 et "infini". la syntaxe "*(element)" signifie donc tout nombre d'occurrences y compris 0; "1*element" précise au moins une occurrence; et "1*2element" précise une ou deux occurrences.

[règle]

Les crochets précisent un élément optionnel; "[foo bar]" vaut pour "*1(foo bar)".

Nrègle

Répétition précise: "<n>(element)" est équivalente à "<n>*<n>(element)"; c'est à dire, exactement <n> occurrences de (element). Ainsi 2DIGIT est un nombre de 2-digits, et 3ALPHA une chaîne de 3 caractères alphabétiques.

#règle

Une syntaxe "#" est utilisée, comme pour la syntaxe "*", pour définir des listes d'éléments. La forme complète en est "<n>#<m>element" indiquant au moins <n> et au plus <m> elements, chacun séparé par une ou plusieurs virgules (",") et des espaces optionnels (LWS). Ceci rend la forme des listes très lisible; une règle du type "(*LWS element *(*LWS "," *LWS element))" peut être vue comme "1#element". Lorsque cette construction est utilisée, les éléments vides sont utilisés, mais ne contribue pas au comptage des éléments présents. De ce fait, "(element), , (element)" est permis, mais compte comme deux éléments. Toutefois, dans les cas ou un élément au moins est requis, cet élément doit être non nul. Les valeurs par défaut sont 0 et "infini" et donc "#(element)" vaut pour tous les éléments y compris 0; "1#element" vaut pour au moins un; et "1#2element" permet un ou deux éléments.

; commentaire

Un point virgule, à distance d'un texte de règle instaure un début de commentaire qui va jusqu'au bout de la ligne. C'est un moyen pratique d'insérer des remarques ou annotations en cours de spécification.

***LWS implicite**

La grammaire décrite par cette spécification est basée sur le "mot". Sauf mention contraire, tout nombre d'espaces (LWS) peut être inséré entre deux mots adjacents ou "token", et entre un "token" et un délimiteur (tspecials), sans changer l'interprétation. Il doit exister au moins un délimiteur (tspecials) entre deux mots, au risque de les voir interprétés comme un seul. Malgré tout, les constructions HTTP tendront à utiliser la "forme commune", certaines implémentations ne savent traiter autre chose que cette forme.

2.2 Règles de base

Les règles de base suivantes seront utilisées tout au long de ce document dans les séquences de recherche. La table de caractères ASCII-US est définie par [17].

```
OCTET                = <toute donnée codée sur 8 bits>
CHAR                 = <tout caractère ASCII-US (0 à 127)>
UPALPHA              = <Tout caractère alphabétique ASCII-US majuscule
                     "A".."Z">
LOALPHA              = <Tout caractère alphabétique ASCII-US minuscule
                     "a".."z">
ALPHA                = UPALPHA | LOALPHA
DIGIT                = <tout digit ASCII-US "0".."9">
CTL                  = <Tous caractère de contrôle ASCII-US (0 à 31) et
                     DEL (127)>
CR                   = <CR ASCII-US, retour chariot (13)>
LF                   = <LF ASCII-US, saut de ligne (10)>
SP                   = <SP ASCCII-US, espace (32)>
HT                   = < HT ASCII-US, tabulation horizontale (9)>
<">                  = <double guillemet ASCII-US (34)>
```

HTTP/1.0 définit la séquence CR LF comme marqueur de fin de ligne pour tous les éléments excepté le corps de l'entité (voir Appendice B pour les tolérances). La fin de ligne à l'intérieur d'un corps d'entité dépend de son média, comme décrit en Section 3.6.

```
CRLF                = CR LF
```

Les en-têtes HTTP/1.0 peuvent être réparties su plusieurs lignes si chaque nouvelle ligne commence par un espace ou une tabulation horizontale. Une suite d'espace, même sur plusieurs lignes équivaut à un espace simple.

```
LWS                 = [CRLF] 1*( SP | HT )
```

Cependant les en-têtes multilignes ne sont pas acceptées par toutes les applications, et doivent de préférence être évitées lors d'un codage HTTP/1.0.

La règle TEXT est utilisée pour décrire des informations descriptives qui ne sont pas sensées être interprétées. Les mots d'un élément *TEXT peuvent contenir d'autres caractères que ceux de la table ASCII-US stricto sensu.

```
TEXT                = <tout OCTET sauf CTL, hormis LWS qui reste
                    autorisé>
```

Les récepteurs d'un élément TEXT d'une en-tête contenant des octets hors de la table ASCII-US supposeront qu'il s'agit de caractères ISO-8859-1.

Les caractères hexadécimaux peuvent être utilisés dans certaines applications.

```
HEX                 = "A" | "B" | "C" | "D" | "E" | "F" | "a" | "b" |
                    "c" | "d" | "e" | "f" | DIGIT
```

De nombreux champs d'en-tête HTTP/1.0 sont formés de mots séparés par des espaces ou caractères spéciaux. Ces caractères doivent être entre guillemets lorsqu'ils sont utilisés à l'intérieur d'une valeur.

```
mot                 = token | chaîne entre guillemets
token               = 1*<tout CHAR excepté CTLs ou tspecials>
tspecials           = "(" | ")" | "<" | ">" | "@" | "," | ";" | ":" |
                    "\" | <"> | "/" | "[" | "]" | "?" | "=" | "{" |
                    "}" | SP | HT
```

Les commentaires pourront être insérés dans les en-têtes HTTP en les entourant de parenthèses. Les commentaires ne sont permis que dans les champs contenant le mot "comment" dans leur définition. Dans tous les autres champs, les parenthèses sont interprétées comme faisant partie de l'expression.

```
comment             = "(" *( ctext | comment ) ")"
ctext               = <tout TEXT excepté "(" et ")">
```

Une chaîne sera interprétée comme un seul mot si elle est entre double guillemets.

```
quoted-string       = ( <"> *(qdtext) <"> )
```

```
qdtext              = <tout CHAR sauf <"> et CTL, hormis LWS qui est
                    accepté>
```

L'utilisation du backslash ("\") comme séquence d'échappement d'un caractère unique n'est pas permis par le protocole HTTP/1.0

3. Paramètres du protocole

3.1 Numéro de Version

HTTP utilise un schéma de numérotation "<supérieure>.<inférieure>" pour indiquer la version du protocole. Cette technique permet à un émetteur d'envoyer une indication sur sa capacité à comprendre une réponse, plutôt que les fonctionnalités qu'il permet. Les composants du message ne pourront altérer ce numéro de version car ils n'affectent pas le comportement de la communication et ne font qu'étendre les possibilités des champs qui les contiennent. Le nombre <inférieur> est incrémenté lorsque les changements apportés au protocole ajoutent des fonctionnalités ne modifiant pas la réponse de l'algorithme d'interprétation, mais qui peuvent ajouter des nouvelles syntaxes ou des nouvelles fonctionnalités côté émetteur. Le nombre <supérieur> est incrémenté lorsque le format de message est changé.

La version d'un message HTTP est indiqué par un champ HTTP-Version dans la première ligne du message. Si la version n'est pas spécifiée, le récepteur prendra par défaut la version la plus restrictive: HTTP/0.9.

```
HTTP-Version          = "HTTP" "/" 1*DIGIT "." 1*DIGIT
```

Notez que les nombres "supérieure" et "inférieure" doivent être pris comme deux entiers distincts et que chacun peut être amené à prendre une valeur supérieure à un simple digit. Ainsi, HTTP/2.4 est une version plus basse que HTTP/2.13, à son tour plus basse que HTTP/12.3. Des zéros mentionnés en tête de nombre doivent être ignorés par le récepteur, et ne devraient pas être émis par l'émetteur.

Ce document définit les versions 0.9 et 1.0 du protocole HTTP. Les applications envoyant des "requêtes pleines" ou "réponses pleines", et définies dans cette spécification, doivent signaler une version HTTP-Version "HTTP/1.0".

Les serveurs HTTP/1.0 doivent:

- reconnaître le format de la ligne de requête HTTP/0.9 et HTTP/1.0
- comprendre toute requête émise sous les formats HTTP/0.9 ou HTTP/1.0;
- répondre avec un message sur la même version de protocole que celle émise dans la requête client.

Les clients HTTP/1.0 doivent:

- reconnaître le format de ligne d'état des réponses HTTP/1.0;
- comprendre toute réponse au format HTTP/0.9 ou HTTP/1.0.

Les applications des Proxy et routeurs doivent prendre certaines précautions lorsqu'ils retransmettent des requêtes d'un format différent que celui de l'application native. Comme la version de protocole indique les possibilités fonctionnelles de l'application de l'émetteur, un proxy/routeur ne doit jamais émettre de requête de niveau supérieur à celui de sa propre application HTTP; si une telle requête lui parvient, le proxy/routeur doit dégrader le numéro de version, ou renvoyer une erreur. Les requêtes reçues sur un niveau inférieur peuvent être relevées au niveau de version de l'application native; la réponse de ces proxies/routeurs doivent néanmoins suivre les règles énoncées ci-avant.

3.2 Uniform Resource Identifiers

URI sont connues sous de nombreux noms: adresses WWW, identificateur universel de document (Universal Document Identifiers), identificateur universel de ressource (Universal Resource Identifiers) [2], et finalement la combinaison d'une localisation uniforme de ressource (Uniform Resource Locators ou URL) [4] et d'un nom (Name - URN) [16]. Pour autant que le protocole HTTP est concerné, les Uniform Resource Identifiers sont simplement des chaînes formatées identifiant --via un nom, une localisation, ou toute autre caractéristiques -- une ressource réseau.

3.2.1 Syntaxe générale

Les URI dans HTTP peuvent être représentées sous forme absolue, ou relativement à une autre URI connue [9], suivant le contexte. Les deux formes sont différenciées par le fait qu'une URI absolue commence toujours par un schème contenant un nom suivi du caractère ":".

```
URI                 = (URIabsolue | URIrelative ) [ "#" fragment ]

URIabsolue          = scheme ":" *( uchar | reservé )

URIrelative         = chem_res | chem_abs | chem_rel

chem_res            = "//" loc_res [ chem_abs ]
chem_abs            = "/" chem_rel
chem_rel            = [ chemin ] [ ";" params ] [ "?" requête ]

chemin              = fsegment *( "/" segment )
fsegment            = 1*pchar
segment             = *pchar

params              = param *( ";" param )
param               = *( pchar | "/" )

scheme              = 1*( ALPHA | DIGIT | "+" | "-" | "." )
loc_res             = *( pchar | ";" | "?" )
requête             = *( uchar | reservé )
fragment            = *( uchar | reservé )

pchar               = uchar | ":" | "@" | "&" | "=" | "+"
uchar               = nonréservé | echap
nonréservé          = ALPHA | DIGIT | sûr | extra | national
```

```
échap                = "%" HEX HEX
réservé              = ";" | "/" | "?" | ":" | "@" | "&" | "=" | "+"
extra                = "!" | "*" | "'" | "(" | ")" | ","
sûr                  = "$" | "-" | "_" | "."
nonsûr               = CTL | SP | <"> | "#" | "%" | "<" | ">"
national             = <tout OCTET hormis ALPHA, DIGIT, reservé, extra,
                     sûr, et nonsûr>
```

Pour une information définitive sur la syntaxe des URL voir les RFC 1738 [4] et RFC 1808 [9]. La notation BNF inclue des caractères nationaux non permis dans les URL valides telles que spécifiées dans la RFC 1738, car les serveurs HTTP ne sont pas limités à l'ensemble de caractères non réservés pour le codage de la prtie relative du chemin d'accès, et les proxies HTTP peuvent recevoir des requêtes à destinations d'URI autres que celles définies dans la RFC 1738.

3.2.2 URL http

Le schème "http" est utilisé pour localiser une ressource réseau via le protocole HTTP. Cette section définit la syntaxe et la sémantique des URL.

```
http_URL             = "http:" "//" host [ ":" port ] [ chem_abs ]

host                 = <un nom Internet d'ordinateur valide ou une
                     adresse IP (sous forme numérique), comme définie
                     en Section 2.1 de la RFC 1123>

port                 = *DIGIT
```

Si le port est vide ou non spécifié, le port 80 est pris par défaut. La sémantique précise que cette ressource est localisée sur le serveur acceptant les connexions TCP sur ce port et sur cet ordinateur, l'URI de requête de la ressource en est le chemin d'accès absolu *chem_abs*. Si *chem_abs* n'est pas précisé dans l'URL, il doit être remplacé par un "/" lorsque l'URI de requête est utilisée (Section 5.1.2).

Note : Bien que le protocole HTTP soit indépendant de la couche transport, l'URL http identifie la ressource par son adresse TCP, Les ressources non TCP devant être atteintes via un autre schème d'URI.

La forme canonique d'une URL "http" est obtenue en convertissant tous les caractères UPALPHA dans la chaîne "host" par leur équivalent LOALPHA (les noms de host ne tiennent pas compte de la casse), en éludant le descriptif [":" port] si le port vaut 80, et en remplaçant le *chem_abs* vide par un "/".

3.3 Formats de temps et de date

Les applications HTTP/1.0 ont historiquement reconnu trois formats distincts pour la définition de dates et d'unités temporelles:

```
Sun, 06 Nov 1994 08:49:37 GMT   ; RFC 822, mis à jour par la RFC 1123
Sunday, 06-Nov-94 08:49:37 GMT  ; RFC 850, obsolète depuis la RFC 1036
Sun Nov  6 08:49:37 1994        ; Format asctime() du C ANSI
```

Le premier format est préféré pour Internet pour une raison liée à la longueur fixe de ses sous-champs, définis par la RFC 1123 [6] (une mise à jour de la RFC 822 [7]). Le second format est d'usage commun, mais se base sur le format de date tel que défini par la RFC 850 [10] désormais obsolète, et présente l'inconvénient d'un champ d'année sur deux digits. Les clients et serveurs HTTP/1.0 doivent reconnaître les trois formats, même s'ils ne doivent jamais générer le troisième (convention du C).

Note : Les récepteurs de valeurs de date doivent avoir une implémentation robuste et de ce fait être capables de reconnaître des formats de date émanent d'applications non-HTTP, comme il peut leur en parvenir en postant ou récupérant des messages via proxy / routeur de et vers SMTP ou NNTP.

Tout index temporel HTTP/1.0 doit être représenté en temps universel (Universal Time - UT), connu aussi sous l'appellation Greenwich Mean Time (GMT), sans exception possible. Ceci est indiqué dans les deux premiers formats par l'inclusion du mot "GMT", abréviation en trois lettre du code de zone, et doit être supposé comme vrai à la lecture d'un temps asctime().

```
HTTP-date           = rfc1123-date | rfc850-date | asctime-date

rfc1123-date        = jsem "," SP date1 SP temps SP "GMT"
rfc850-date         = joursem "," SP date2 SP temps SP "GMT"
asctime-date        = jsem SP date3 SP temps SP 4DIGIT

date1               = 2DIGIT SP mois SP 4DIGIT
                    ; jour mois an (e.g., 02 Jun 1982)
date2               = 2DIGIT "-" mois "-" 2DIGIT
                    ; jour-mois-an (e.g., 02-Jun-82)
date3               = mois SP ( 2DIGIT | ( SP 1DIGIT ))
                    ; mois jour (e.g., Jun  2)

temps               = 2DIGIT ":" 2DIGIT ":" 2DIGIT
                    ; 00:00:00 - 23:59:59

jsem                = "Mon" | "Tue" | "Wed"
                    | "Thu" | "Fri" | "Sat" | "Sun"

joursem             = "Monday" | "Tuesday" | "Wednesday"
                    | "Thursday" | "Friday" | "Saturday" | "Sunday"

mois                = "Jan" | "Feb" | "Mar" | "Apr"
                    | "May" | "Jun" | "Jul" | "Aug"
                    | "Sep" | "Oct" | "Nov" | "Dec"
```

Note : Les contraintes au niveau du format de date HTTP ne sont valables qu'à l'intérieur du réseau de transmission. Les clients et serveurs ne sont pas obligés de respecter cette présentation pour l'affichage, la présentation de requête, l'accès, etc.

3.4 Tables de caractères

HTTP utilise la même définition du vocable "table de caractères" que celle utilisée par MIME:

Le terme "table de caractères" utilisé dans ce document se réfère à une méthode utilisant une ou plusieurs tables pour convertir une séquence d'octets en caractères. Notez qu'une conversion inverse n'est pas forcément demandée, c'est à dire que tout octet peut ne pas être représenté par un caractère, et qu'à l'inverse, une chaîne de caractères peut très bien être représentable par plus d'une seule séquence d'octets. Cette définition permet un grand nombre de types de conversions, depuis celle, élémentaire, à une table comme l'ASCII-US jusqu'à des méthodes plus complexes à "commutation de tables" comme celles définies par l'ISO 2022. Cependant, la définition associée au nom d'une "table de caractère" MIME doit parfaitement et entièrement spécifier le passage des octets aux caractères. En particulier, le recours à des informations externes pour pouvoir assumer la conversion est interdit.

Note : Le terme "table de caractères" est aussi communément appelé "encodage". Cependant, comme HTTP et MIME partagent les mêmes définitions, il est important qu'ils partagent aussi la terminologie.

Les "tables de caractères" HTTP sont définies par des mots sans distinction de casse. L'ensemble complet de ces tables constitue le "registre de tables de l'IANA" (Character Set registry [15]). Cependant, comme ce "registre" ne définit pas un nom unique et définitif pour chacune de ces tables, nous avons défini ici les noms les plus couramment utilisés dans les entités HTTP. Ces "tables de caractères" incluent celles définies dans la RFC 1521 [5] -- ASCII-US [17] et ISO-8859 [18] -- ainsi que celles spécifiquement définies pour usage dans les paramètres de caractères MIME.

```
charset              = "US-ASCII" | "ISO-8859-1" | "ISO-8859-2"
                     | "ISO-8859-3" | "ISO-8859-4" | "ISO-8859-5"
                     | "ISO-8859-6" | "ISO-8859-7" | "ISO-8859-8"
                     | "ISO-8859-9" | "ISO-2022-JP" | "ISO-2022-JP-2"
                     | "ISO-2022-KR" | "UNICODE-1-1" | "UNICODE-1-1-
                     UTF-7"
                     | "UNICODE-1-1-UTF-8" | autre_nom
```

Bien qu'HTTP permette l'utilisation de tout nom arbitraire pour désigner une table de caractères, tout nom désignant déjà une des tables du registre IANA doit indiquer cette table précise. Il est conseillé que les applications s'en tiennent aux tables de caractères définies dans ce registre.

La table de caractère indiquée pour un corps d'entité doit être le plus petit dénominateur commun de l'ensemble des codes de caractères utilisés dans ce corps, sauf si l'on préfère systématiquement s'en tenir aux tables de base ASCII-US ou ISO-8859-1.

3.5 Indication d'encodage du contenu

Les valeurs de "codage de contenu" indiquent la nature de la transformation qui a été appliquée à une ressource. Cette information a été initialement définie pour pouvoir garder la trace d'un type de ressource compressée ou encryptée. Typiquement, la ressource est disponible sous forme encodée, et ne sera décodée qu'au moment de son utilisation.

```
content-coding              = "x-gzip" | "x-compress" | transformation
```

Note : Pour des raisons de compatibilité future, les applications HTTP/1.0 doivent interpréter "gzip" et "compress" respectivement comme "x-gzip" et "x-compress".

Toute mention d'encodage est indépendante de la casse. L'HTTP/1.0 utilise les descripteurs d'encodage dans le champ Content-Encoding (Section 10.3) de l'en-tête. Le plus important est que cette information indique quel est le type de décodage que le récepteur devra utiliser pour exploiter la ressource. Notez qu'un même programme peut être capable de décoder plusieurs types d'encodage. Les deux valeurs définies dans cette spécification sont:

x-gzip
Un format de compression géré par le programme de compression "gzip" (GNU zip) développé par Jean-Loup Gailly. Ce codage est de type Lempel-Ziv (LZ77) avec CRC sur 32 bits.

x-compress
Un format de compression géré par le programme "compress" effectuant une compression adaptative de type Lempel-Ziv-Welch (LZW).

Note : L'utilisation de noms de programmes pour nommer des formats d'encodage n'est pas souhaitable et sera déconseillé dans le futur. Leur utilisation dans ce cas provient surtout d'une habitude historique, mais ce n'est pas une habitude à prendre.

3.6 Types de média

HTTP exploite les définition de types de média Internet Media Types [13] dans le champ Content-Type de l'en-tête (Section 10.5) afin de proposer une méthode de typage ouverte et évolutive.

```
media-type          = type "/" soustype *( ";" paramètre )
type                = nom_type
soustype            = nom_soustype
```

Les paramètres suivent la description de type/soustype, sous la forme de paires attribut/valeur.

```
paramètre           = attribut "=" valeur
attribut            = nom_attribut
valeur              = nom_valeur | chaîne ente guillemets
```

Le type, sous-type et les noms d'attributs sont insensibles à la casse. Les valeurs associées aux attributs peuvent l'être ou ne pas l'être, suivant la sémantique du nom de l'attribut. les types et sous type ne peuvent être séparé par un LWS. Ceci est valable pour les paires attribut/valeur Lorsqu'un récepteur reçoit un paramètre irrecevable pour le type de média spécifié, il devra traiter le descripteur comme si cette paire attribut/valeur n'existait pas..

Certaines anciennes applications HTTP ne reconnaissent pas le type de média. Les applications HTTP/1.0 ne pourront définir le type que lorsque la spécification du contenu est indispensable.

Les valeurs admises de types de média sont définies par la Internet Assigned Number Authority (IANA [15]). Le processus d'enregistrement d'un type de média est spécifié dans la RFC 1590 [13]. L'usage de types non enregistrés est déconseillé.

3.6.1 Canonisation et texte par défaut

Les types de média Internet sont enregistrés sous une forme canonique. En principe, un corps d'entité transféré par HTTP doit être représenté dans sa forme canonique avant transmission. Si le corps a été encodé sous un codage spécifié par un Content-Encoding, il devait être sous sa forme canonique avant encodage.

Les sous types du type "text" utilisent dans leur forme canonique la séquence CRLF en tant que fin de ligne. Cependant, HTTP permet le transport de texte dans lequel un CR ou un LF seul représente la fin de ligne, tant que l'on reste à l'intérieur du corps de l'entité. Les applications HTTP doivent accepter des fins de ligne représentées par la séquence CRLF, par un CR seul, ou par un LF seul, dès qu'il s'agit d'un média "text".

De plus, si le média texte utilise une table de caractère dans laquelle les caractères 13 et 10 ne correspondent pas aux caractères CR et LF, comme c'est le cas pour certains codages multi-octets, HTTP permet l'usage de toute séquence de caractères désignés comme équivalents des CR et LF et faisant office de fins de ligne. Cette souplesse n'est permise par HTTP que dans le corps de l'entité; Un CRLF "légal" ne devra pas être remplacé par un CR seul ou un LF seul dans aucune des structures de contrôle HTTP (telles que les champs d'en-têtes ou les limites "multipart").

Le paramètre "charset" est utilisé avec certains types de média pour définir la table de caractères utilisée (Section 3.4) dans les données. Lorsque ce paramètre n'est pas explicitement renseigné, les sous-types du média "text" prennent par défaut la table de caractères "ISO-8859-1". Des données codées selon une autre table de caractères que "ISO-8859-1" ou ses sous-ensembles devra nécessairement entraîner une spécification explicite de ce codage, sous peine d'être ininterprétables par le récepteur.

Note : De nombreux serveurs HTTP actuels diffusent des données codées selon une table de caractères autre que "ISO-8859-1" et sans explicitation correcte. Cette attitude réduit l'interopérabilité propre du réseau et est fortement déconseillée. Pour compenser de manque de rigueur, certains utilisateurs HTTP proposent une option permettant de changer la table de caractères par défaut utilisée lorsque des données sont reçues sans explicitation de cette information.

3.6.2 Types multiples "multipart"

MIME définit un certain nombre de types "multipart" -- encapsulation de plusieurs entités dans un même corps d'entité. Les types "multipart" enregistrés par l'IANA [15] n'ont pas de signification particulière aux yeux d'HTTP/1.0, bien que les utilisateurs dussent être en mesure de pouvoir reconnaître chaque sous type de façon à correctement interpréter chaque sous partie du corps. Un utilisateur HTTP suivra les mêmes règles de comportement qu'un utilisateur MIME dans le cas d'une réception de données "multipart". Les serveurs HTTP ne sont pas sensé supposer que tous les clients HTTP puissent recevoir des données ainsi présentées.

Tous les types "multipart" partagent la même syntaxe et doivent spécifier un paramètres de "limite" dans la définition du type. Le corps du message fait lui-même partie du protocole et doit s'en tenir à la séquence CRLF pour représenter des sauts de

lignes entre les parties. Chaque partie d'un document "multipart" peut elle même contenir des champs d'en-tête HTTP significatifs pour son contenu.

3.7 Produits

La spécification de produit permet à deux application en communication de s'identifier l'une à l'autre par un simple nom, suivi d'un "slash" ('/') et d'un numéro de version, tous deux optionnels. Les champs acceptant cette spécification permettent de lister des sous-éléments significatifs d'un produit, séparés par un espace. Par convention, les produits sont rangés par ordre d'importance dans le processus d'identification.

```
produit              = nom_produit ["/" version]
version              = numéro_de_version
```

Exemples:

```
User-Agent: CERN-LineMode/2.15 libwww/2.17b3

Server: Apache/0.8.4
```

Les spécifications de produits doivent être concises et directes -- l'utilisation de ce champ à vocation publicitaire ou pour diffuser des informations non essentielles est strictement interdit. Le sous-champ de numéro de version accepte tout caractère valide pour des noms. Il est demandé de ne pas abuser de cette permissivité et de conserver ce champ pour l'indication de la version à l'exclusion de toute autre information.

4. Messages HTTP

4.1 Types de messages

Les messages HTTP consistent en des requêtes émises par un utilisateur à destination d'un serveur, ou d'une réponse d'un serveur au destinataire.

```
HTTP-message          = Requête_simple              ; HTTP/0.9 messages
                      | Réponse_simple
                      | Requête_complète            ; HTTP/1.0 messages
                      | Réponse_complète
```

Les requêtes et réponses "complètes" utilisent le format de message défini dans la RFC 822 [7] concernant la transmission d'entités. Ces deux messages peuvent contenir une en-tête optionnelle ainsi que le corps de l'entité. Le corps et l'en-tête de l'entité sont séparés par une ligne vide (soit une séquence CRLFCRLF).

```
Requête_complète      = Ligne_requête             ; Section 5.1
                      *( En-tête_générale         ; Section 4.3
                      | En-tête_requête           ; Section 5.2
                      | En-tête_entité )          ; Section 7.1
                      CRLF
                      [ Corps_entité ]            ; Section 7.2

Réponse_complète      = Ligne-état                ; Section 6.1
                      *( En-tête_générale         ; Section 4.3
                      | En-tête_réponse           ; Section 6.2
                      | En-tête_entité )          ; Section 7.1
                      CRLF
                      [ Corps_entité ]            ; Section 7.2
```

Les requête et réponse simple ne transportent jamais d'en-tête. Une seule requête est de ce type : GET.

```
Requête_simple        = "GET" SP URI-visée CRLF

Réponse_simple        = [ Corps_entité ]
```

L'usage des requêtes simples reste déconseillé, car il ne permet pas au serveur de définir le type de média dans sa réponse.

4.2 En-têtes de messages

Les champs d'en-têtes HTTP, dont l'*en-tête_générale* (Section 4.3), l'*en-tête_requête* (Section 5.2), L'*en-tête_réponse* (Section 6.2), et l'*en-tête_entité* (Section 7.1), ont tous le même format générique décrit dans le paragraphe 3.1 de la RFC 822 [7]. Chaque champ d'en-tête consiste en un nom suivi immédiatement d'un deux-points (":"), un espace (SP), et la valeur du paramètre. Les noms de champs sont insensibles à la casse. Les champs d'en-tête peuvent être écrits sur plusieurs lignes, à condition que le premier caractères des lignes suivantes soit SP ou HT. Cette pratique reste cependant déconseillée.

```
HTTP-header          = nom ":" SP [ valeur ] CRLF

nom                  = nom_de_champ
valeur               = *( contenu | LWS )

contenu              = <les OCTETs décrivant la valeur, pouvant être un
                     *TEXT ou combinaison de noms, caractères spéciaux,
                     et chaînes entre guillemets>
```

L'ordre dans lequel les champs d'en-tête sont reçus n'a pas d'importance. Une "bonne habitude" consistera toutefois à envoyer les champs d'en-tête_générale en premier, suivi des champs d'en-tête_requête ou d'en-tête_réponse, puis enfin les champs d'en-tête_entité.

Lorsque plusieurs champs de même nom doivent être définis avec des valeurs différentes, celles-ci doivent apparaître comme une liste séparée par des virgules [#(values)]. Il doit être possible de combiner cette définition multiple à partir des paires individuelles "nom: valeur", sans changer la sémantique du message, en ajoutant chaque valeur à la suite de la première, en utilisant une virgule comme séparateur.

4.3 En tête générale

Certains champs d'en-tête ont une utilité aussi bien dans des requêtes que dans des réponses, en conservant la même signification. Les informations définies dans ces champs concernent le message lui-même, et non l'entité qu'il transporte.

```
General-Header       = Date              ; Section 10.6
                     | Pragma            ; Section 10.12
```

Des nouveaux noms de champs d'en-tête_générale ne peuvent être introduits que dans le cadre d'un changement de version du protocole. Cependant, de nouveaux champs ou champs expérimentaux peuvent utiliser la sémantique de champs d'en-tête_générale, à partir du moment ou les deux extrémités de la communication sont d'accord pour le faire. Tout champ non reconnu sera considéré comme un champ d'en-tête_entité.

5. Requêtes

Une requête d'un client vers un serveur inclue, dans sa première ligne, la méthode appliquée à la ressource, l'identificateur de cette ressource, et la version de protocole courante. Afin d'assurer la compatibilité descendante avec la version plus limitée HTTP/0.9 du protocole, deux formats seront valides pour exprimer une requête:

```
Request              = Requête_simple | Requête_complète

Simple-Request       = "GET" SP URI-visée CRLF

Requête_complète     = Ligne-requête           ; Section 5.1
                     *( En-tête_générale       ; Section 4.3
                     | En-tête_requête         ; Section 5.2
                     | En-tête_entité )        ; Section 7.1
                     CRLF
                     [ Corps-entité ]          ; Section 7.2
```

Si un serveur HTTP/1.0 reçoit une requête simple, il devra répondre par une réponse simple HTTP/0.9. Un client HTTP/1.0 capable de recevoir une réponse_complète ne doit jamais générer de requête_simple.

5.1 Ligne de requête

La ligne de requête commence par un nom de requête, suivie de l'URI visée, du numéro de version de protocole, et termine enfin par CRLF. Ces éléments sont séparés par des espaces. Aucun CR ni LF n'est autorisé excepté la séquence finale CRLF.

```
Ligne_requête        = Méthode SP URI-visée SP Version-HTTP CRLF
```

Notez que la différence entre une ligne de requête_simple et de requête_complète ne réside que dans la présence du numéro de version HTTP, et dans la possibilité d'appeler d'autres méthodes que GET.

5.1.1 Méthodes

La méthode indiquée en tête de ligne est destinée à être appliquée à l'URI cible. Son nom dépend de la casse.

```
Méthode              = "GET"                   ; Section 8.1
```

```
                | "HEAD"                  ; Section 8.2
                | "POST"                  ; Section 8.3
                | nom_de_méthode
```

La liste de méthodes que peut accepter une ressource est dynamique; Le client est avisé de la disponibilité de la méthode émise par le code de retour du message de réponse. Les serveurs renverront le code 501 (non implémenté) si la méthode est inconnue, ou non implémentée.

Les méthodes habituellement employées par les applications HTTP/1.0 sont décrites en détail en Section 8.

5.1.2 URI-visée (Request-URI)

L'URI visée est l'URI identifiant la ressource réseau à laquelle doit être appliquée la méthode.

```
URI-visée           = URIabsolue | chem_abs
```

Ces deux options dépendent de la méthode appelée.

Seule l'option URIabsolue est permise lorsque la requête est envoyée à un proxy. Le proxy devra retransmettre la requête, et servir d'intermédiaire pour la réponse. Si la requête est GET ou HEAD et la réponse présente dans le cache du proxy, celui-ci pourra utiliser cette réponse directement, si les conditions restrictives de "date d'expiration " dans l'en-tête sont remplies. Notez que le proxy peut à son tour retransmettre la requête vers un autre proxy, ou directement vers le serveur origine. Afin d'éviter qu'une requête ne boucle, un proxy doit reconnaitre tous sesnoms de serveurs, y compris alias, variations locales, et les adresses IP numériques. Voici un exemple de ligne de requête:

```
GET http://www.w3.org/pub/WWW/TheProject.html HTTP/1.0
```

La forme d'URI-visée la plus commune est celle qui identifie une ressource sur son serveur origine ou un routeur. Dans ce cas, seul le chemin absolu *chem_abs* est transmis (Cf. Section 3.2.1, chem_abs). Par exemple, un client souhaitant récupérer la ressource ci-dessus directement à partir de son serveur origine créera une connexion TCP sur le port 80 du host "www.w3.org" et émettra la ligne de commande:

```
GET /pub/WWW/TheProject.html HTTP/1.0
```

suivi du reste de la requête complète. Notez qe le chemin absolu ne peut être vide; si la ressource se trouve dans la racine (pas de chemin d'accès) le chemin spécifié devra comporter au moins le caractère slash ("/").

L'URI-visée est transmise sous forme d'une chaîne encodée, dans laquelle certains caractères apparaîtront comme une séquence d'échappement "% HEX HEX" telle que définie par la RFC 1738 [4]. Le serveur origine devra décoder cette chaîne afin d'accéder correctement à la ressource.

5.2 En-tête de requête

Les champs d'en-tête de requête permettent de la transmission d'informations complémentaires sur la requête, et le client lui-même. Ces champs agissent comme "modificateurs" de la requête, utilisant une sémantique identique à celle des paramètres passés par un appel d'une méthode de langage de programmation de haut niveau.

```
Request-Header       = Authorization            ; Section 10.2
                     | From                     ; Section 10.8
                     | If-modified-since        ; Section 10.9
                     | Referer                  ; Section 10.13
                     | User-agent               ; Section 10.15
```

Des nouveaux noms de champs d'en-tête_requête ne peuvent être introduits que dans le cadre d'un changement de version du protocole. Cependant, de nouveaux champs ou champs expérimentaux peuvent utiliser la sémantique de champs d'en-tête_requête, à partir du moment ou les deux extrémités de la communication sont d'accord pour le faire. Tout champ non reconnu sera considéré comme un champ d'en-tête_entité.

6. Réponse

Une fois la requête reçue et interprétée, un serveur répond par un message HTTP de réponse.

```
Réponse              = Réponse_simple | Réponse_complète

Réponse_simple       = [ Corps_entité ]

Réponse_complète     = Ligne_état             ; Section 6.1
                     *( En-tête_générale      ; Section 4.3
                     | En-tête_réponse        ; Section 6.2
                     | En-tête_entité )       ; Section 7.1
                     CRLF
                     [ Corps_entité ]         ; Section 7.2
```

Une réponse_simple ne peut être envoyé qu'en réponse d'une requête_simple HTTP/0.9 ou si le serveur ne supporte que la version limitée de protocole HTTP/0.9. Si un client émet une requête_complète HTTP/1.0 et reçoit une réponse ne commençant pas par une ligne_état, il devra conclure qu'il s'agit d'une réponse_simple et l'interprétera en conséquence. Notez qu'une réponse ne contient que le corps_entité, et se termine par la fermeture de la connexion par le serveur.

6.1 Ligne d'état

La première ligne d'un message de réponse_complète est la ligne d'état, constituée d'une indication du numéro de version du protocole, suivi du code numérique de la réponse, suivi enfin d'une explicitation textuelle de cette réponse, chaque élément étant séparé par un espace. Aucun CR ni LF ne peuvent y apparaître à l'exception de la séquence CRLF finale.

```
Ligne_état           = Version_HTTP SP Code_état SP Raison CRLF
```

La ligne d'état débutant toujours par ce numéro de version et le code de réponse:

```
"HTTP/" 1*DIGIT "." 1*DIGIT SP 3DIGIT SP
```

(ex., "HTTP/1.0 200 "), la seule présence de cette expression est suffisante pour différentier une réponse_simple d'une réponse_complète émise en retour d'une requête

sousle protocole HTTP/1.0. Le format de réponse_simple n'interdit cependant pas qu'une telle expression soit placée en tête du corps d'entité, provoquant ainsi une erreur d'interprétation de la part du récepteur. Dans la pratique, la plupart des serveurs HTTP/0.9 ne peuvent générer que des réponses de type "text/html", ce qui en principe, élimine le risque d'une telle confusion.

6.1.1 Code d'état et raison explicite

L'élément code_état est un nombre entier à 3 chiffres indiquant le succès ou la cause d'erreur de la transaction. L'élément "raison" est un commentaire textuel destiné à identifier explicitement la cause d'erreur. Le code d'état sera en général exploité par des automates. La raison est à destination de notre intellect humain.
Celle-ci n'est pas obligatoirement traitée ni reportée par le client.

Le premier chiffre du code d'état indique la classe générale de la réponse. Les deux derniers n'ont pas de rôle de classification particulier. Le premier chiffre peut prendre 5 valeurs:

- **1xx: Information** - Non utilisé, pour usage futur
- **2xx: Succès** - L'action a été correctement reçue, interprétée, et exécutée.
- **3xx: Redirection** - Une décision supplémentaire doit être prise pour terminer la requête
- **4xx: Erreur Client** - La requête présente une erreur de forme et ne peut être satisfaite
- **5xx: Erreur Serveur** - La requête est valide, mais le serveur ne peut la satisfaire

Les valeurs actuellement reconnues sous HTTP/1.0, et les phrases types d'explicitation associées, sont présentées ci-dessous. Ces phrases ne sont que recommandées -- elles peuvent être remplacées par des variations locales sans affecter le protocole. Ces codes sont intégralement listés en section 9.

```
Code_état   = "200" ; OK                    OK
            | "201" ; Created               Créé
            | "202" ; Accepted              Accepté
            | "204" ; No Content            Pas de contenu
            | "301" ; Moved Permanently     Changement définitif
            | "302" ; Moved Temporarily     Changement temporaire
            | "304" ; Not Modified          Non modifié
            | "400" ; Bad Request           Requête incorrecte
            | "401" ; Unauthorized          Non autorisé
            | "403" ; Forbidden             Interdit
            | "404" ; Not Found             Non trouvé
            | "500" ; Internal Server Error Erreur interne serveur
            | "501" ; Not Implemented       Non implémenté
            | "502" ; Bad Gateway           Erreur de routeur
            | "503" ; Service Unavailable   Indisponible
            | autres_codes

autres_codes         = 3DIGIT

Raison               = *<TEXT, sauf CR, LF>
```

La liste des codes HTTP est extensible, mais seuls les codes ci-dessus sont utilisés dans la pratique courante. Les applications HTTP n'ont pas nécessairement à connaître la signification de tous les codes enregistrés, bien que ce soit souhaitable pour des raisons évidentes. Au minimum, les applications devront pouvoir comprendre le premier chiffre de ce code, et comprendre tout numéro de réponse non identifié comme le numéro x00 de la classe, indiquant ainsi la définition générique de la classe. Aucune réponse non identifiée ne peut être enregistrée en cache.
Par exemple, si un code "inconnu" 431 est reçu par le client, celui-ci peut interpréter sans doute possible qu'un problème est survenu, et peut réagir comme s'il avait reçu le code 400. Dans de tels cas, il est fortement conseillé que l'application reporte le corps de l'entité de réponse, dans la mesure où il y a de fortes chances que celui-ci contienne des informations "en clair" sur la nature de l'erreur.

6.2 En-tête de réponse

Les champs d'en-tête de réponse véhiculent certaines informations complémentaires concernant cette réponse, et qui ne peuvent être mentionnées dans la ligne d'état. Ces champs donnent des informations sur le serveur lui-même, et sur les actions à mener par la suite pour accéder à la ressource demandée.

```
Response-Header      = Location                 ; Section 10.11
                     | Server                   ; Section 10.14
                     | WWW-Authentificate       ; Section 10.16
```

Des nouveaux noms de champs d'en-tête_réponse ne peuvent être introduits que dans le cadre d'un changement de version du protocole. Cependant, de nouveaux champs ou champs expérimentaux peuvent utiliser la sémantique de champs d'en-tête_réponse, à partir du moment ou les deux extrémités de la communication sont d'accord pour le faire. Tout champ non reconnu sera considéré comme un champ d'en-tête_entité.

7. Entités

Les messages de requête et de réponses contiennent généralement une entité dans laquelle est incluse des éléments de requête ou de réponse. Une entité est définie par une en-tête d'entité, et dans la plupart des cas par un corps d'entité. Dans ce qui suit "l'émetteur" et le "récepteur" se réfèrent tantôt au client, tantôt au serveur, suivant l'agent qui émet le message.

7.1 En-tête d'entité

Les champs d'en-tête d'entité véhiculent certaines métainformations concernant le corps d'entité, ou, si celui-ci n'existe pas, la ressource visée par la requête.

```
En-tête_entité       = Allow                    ; Section 10.1
                     | Content-Encoding         ; Section 10.3
                     | Content-Length           ; Section 10.4
                     | Content-Type             ; Section 10.5
                     | Expires                  ; Section 10.7
                     | Last-Modified            ; Section 10.10
                     | champ_en-tête

champ_en-tête        = en-tête_HTTP
```

Le mécanisme d'extension de l'en-tête d'entité permet la définition d'autres champs, ceux-ci n'étant pas obligatoirement reconnus par le récepteur. Tout champ non identifié doit être ignoré, ou, dans le cas d'un proxy, retransmis tel quel.

7.2 Corps d'entité

Le corps d'entité (s'il existe) envoyé dans un message de requête ou de réponse HTTP est dans un format et sous un encodage défini par les champs d'en-tête d'entité.

```
Corps_entité         = *OCTET
```

Un corps d'entité n'apparaît dans un message de requête que dans la mesure ou le type de requête le demande. La présence de ce corps est signalée par la présence d'un champ Content-Length dans les champs d'en-tête de la requête. Les requêtes HTTP/1.0 contenant un corps doivent nécessairement présenter un champ d'en-tête Content-Length valide.

Dans les réponses, la présence d'un corps d'entité est fonction à la fois de la nature de la requête préalable, et du code d'état renvoyé. Toute réponse à une requête de type HEAD ne peut contenir de corps d'entité, même si les champs d'en-tête présents prétendent le contraire. Les codes d'état 1xx (informations), 204 (pas de contenu), et 304 (non modifié) supposent implicitement ou explicitement l'absence de tout corps d'entité. Toutes les autres réponses peuvent contenir un corps d'entité, ou à défaut, un champ Content-Length spécifié à zéro (0).

7.2.1 Type

Lorsqu'un corps d'entité est présent dans le message, le type de données incluses dans ce corps est précisé par les champs d'en-tête Content-Type et Content-Encoding. Ceux-ci définissent un modèle d'encodage arborescent à deux niveaux:

```
Corps_entité          = Content-Encoding( Content-Type( données ) )
```

Le Content-Type le type de média des données de la ressource. Un champ Content-Encoding peut être utilisé pour spécifier un traitement ou encodage supplémentaire effectué sur le type de données, souvent dans un but de compression, et apparaissant comme une propriété de la ressource. Par défaut, aucun encodage n'est appliqué aux données (la ressource est enregistrée et directement accessible sous forme utilisable).

Tout message HTTP/1.0 contenant un corps d'entité doit au moins faire apparaître le champ Content-Type définissant la nature des données de la ressource. Si et seulement si aucun type de média n'est défini (uniquement dans le cas où le champ Content-Type n'existe pas, comme dans le cas de réponses simples), le récepteur pourra être amené à déterminer ce type par lui-même, en inspectant le contenu des données, ou en se basant sur l'extension utilisée dans l'URL définissant l'accès à la ressource. Si le média demeure non identifiable, le type par défaut "application/octet-stream".

7.2.2 Longueur

Lorsqu'un corps d'entité est présent dans un message, la longueur de ce corps doit être explicitée par l'un des deux moyens suivants. Si un champ Content-Length est présent, sa valeur associée représente la longueur en octets du corps d'entité.
Autrement, c'est la déconnexion par le serveur qui marque la fin du corps d'entité.

Cette dernière méthode ne peut être utilisée pour marquer la fin d'une requête, car la possibilité doit être laissée au serveur d'envoyer une réponse. C'est pourquoi le protocole HTTP/1.0 précise que toute requête contenant un corps d'entité doit mentionner sa longueur dans un champ d'en-tête Content-Length valide. Si tet est le cas, et que ce champ n'existe pas dans le message, et dans la mesure où le serveur ne peut calculer ou déterminer la longueur de ce corps, ce dernier répondra par un message de code 400 (erreur client).

Note : Certains anciens serveurs spécifient un champ Content-Length erroné lorsque la réponse contient des données dynamiquement générées par le serveur. Il est signalé ici que cet état de fait ne sera plus tolérable dans les nouvelles versions d'HTTP.

8. Définition des méthodes

L'ensemble des méthodes courantes d'HTTP/1.0 est défini ci-dessous. Bien que cet ensemble puisse être complété, il est précisé ici que des méthodes additionnelles implémentées par des clients ou serveurs distincts ne peuvent partager la même sémantique que si leur fonction est identique.

8.1 GET

La méthode GET signifie "récupérer" le contenu quel qu'il soit de la ressource (sous forme d'une entité) identifiée par l'URI-visée. Si l'URI-visée identifie un processus générant dynamiquement des données, ce sont les données produites qui sont renvoyées dans l'entité au lieu du source de l'exécutable appelé, sauf si ce texte lui-même est la sortie du processus.

La sémantique de la méthode GET introduit une notion conditionnelle si la requête inclue le champ d'en-tête If-Modified-Since. Une méthode GET conditionnelle ne récupère la ressource que si celle-ci a été modifiée depuis la date associée au champ If-Modified-Since, comme indiqué en Section 10.9. La méthode GET conditionnelle est utilisée pour réduire la charge du réseau en permettant à des ressources en cache d'être rafraîchies sans multiplier les requêtes ou transférer des données inutiles.

8.2 HEAD

La méthode HEAD est identique à la méthode GET, sauf qu'elle ne demande pas la transmission du corps d'entité de la ressource en retour. Seule les métainformations constituant l'en-tête HTTP de la ressource sont envoyées. Cette méthode permet de ne demander que les informations concernant la ressource (au sens d'HTTP). Elle est utilisée à des fins de tests d'hyperliens, vérification d'existence ou d'actualité d'une ressource.

Il n'existe pas de méthode HEAD conditionnelle comme pour la méthode GET. Si un champ d'en-tête If-Modified-Since est présent dans une requête HEAD, il devra être ignoré.

8.3 POST

La méthode POST est utilisée pour indiquer au serveur de soumettre l'entité contenue dans le message à la ressource identifiée par l'URI-visée. POST est destinée à fournir un moyen uniforme pour les opérations suivantes:

- Annotation de ressources existantes;
- Envoi d'un message vers une édition en ligne, un groupe de nouvelles, une liste d'adresse, ou listes similaires;
- Fournir un bloc de données, par exemple, un résultat de formulaire [3], à un processus de gestion de données;
- Ajout d'éléments à une base de données.

La fonction effective de la méthode POST est définie par le serveur et dépend de l'URI désignée. L'entité "postée" est subordonnée à cette URI dans le même sens qu'un fichier est subordonné au répertoire qui le contient, un article est subordonné au groupe de nouvelles à qui il a été posté, ou un enregistrement à une base de données.

La résolution d'une méthode POST ne signifie pas forcément la création d'une entité sur le serveur origine, ni une possibilité d'accès futur à ces informations. En d'autres termes, le résultat d'un POST n'est pas nécessairement une ressource associable à une URI. Dans ce cas, une réponse de code 200 (OK) ou 204 (pas de contenu) est la réponse la plus appropriée, suivant que la réponse contient ou non une entité pour décrire le résultat.

Si une ressource a été créée sur le serveur origine, la réponse sera du type 201 (créé) et contiendra l'entité (de préférence du type "text/html") qui décrira les informations sur la requête et contiendra une référence sur la nouvelle ressource.

Un champ Content-Length valide est demandé dans toute requête POST HTTP/1.0. Un serveur HTTP/1.0 répondra par un message 400 (requête incorrecte) s'il ne peut déterminer la longueur du contenu du message.

Les applications ne peuvent enregistrer en cache les réponses à des requêtes POST dans la mesure où il n'est absolument pas certain qu'une réponse ultérieure émise dans les mêmes conditions produise le même résultat.

9. Définition des codes d'état

Tous les codes d'état actuellement valides sont décrits ci-dessous, en précisant quelle méthode peut en être l'origine, ainsi que les métainformations à fournir dans la réponse.

9.1 Information 1xx

Cette classe de réponse est actuellement réservée pour de futures applications, et consiste en des messages avec une ligne d'état, des champs d'en-têtes éventuels, et terminés par une ligne vide (CRLFCRLF).
HTTP/1.0 ne définit actuellement aucun de ces codes, lesquels ne constituent pas une réponse valide à des requêtes HTTP/1.0. Il restent cependant exploitables à titre expérimental, dépassant le contexte de la présente spécification.

9.2 Succès 2xx

Cette classe précise que la requête du client a été correctement transmise, interprétée, et exécutée.

200 OK

La requête a abouti. L'information retournée en réponse dépend de la requête émise, comme suit:

GET

Une entité correspondant à l'URI visée par la requête est renvoyée au client;

HEAD

La réponse au client ne doit contenir que les champs d'en-tête à l'xclusion de tout corps d'entité;

POST

Une entité décrivant le résultat de l'action entreprise.

201 Créé

La requête a abouti, et une nouvelle ressource réseau en résulte.

La nouvelle ressource créée est accessible par l'URI(s) renvoyée dans l'entité de la réponse. La ressource doit être créée avant que ce code ne soit envoyé. Si la création ne

peut pas être opérée immédiatement, le serveur devra indiquer dans la réponse quand la ressource deviendra disponible ou retourner un message de type 202 (acceptée).

Actuellement, seule la méthode POST peut conduire à la création d'une ressource.

202 Acceptée

La requête a été reçue et interprétée correctement, mais son traitement n'est pas terminé. La requête peut ou ne peut être réémise pendant le traitement, suivant que le serveur autorise ou n'autorise pas un arrêt du processus en cours. Il n'est pas possible de réémettre un code d'état dans ce type d'opération asynchrone.

La réponse 202 est intentionnellement non bloquante. Son rôle est de permettre au serveur d'accepter une requête d'un autre processus (par exemple d'un automate se déclenchant à dates fixes) sans que la connexion avec le client ne reste active pendant le déroulement du traitement. L'entité retournée par la réponse rendra compte de la requête, et devra fournir un pointeur sur un composant d'information (état courant du traitement), ou donner une estimation de la date de conclusion probable définitive.

204 Pas de contenu

La requête a abouti et le serveur n'a rien à envoyer en retour.

Un utilisateur recevant ce message n'aura pas à rafraîchir son affichage, et maintiendra celui qui aura conduit à l'émission de la requête. Cette réponse a été instaurée pour permettre à un utilisateur de dérouler des scripts ou autres actions sans modifier l'affichage en cours. La réponse peut cependant contenir des métainformations dans des champs d'en-tête, concernant le document affiché dans la fenêtre active de l'utilisateur.

9.3 Redirection 3xx

Cette classe de messages précise que le client doit provoquer une action complémentaire pour que la requête puisse être conduite jusqu'à sa résolution finale. L'action peut être déclenchée par l'utilisateur final si et seulement si la méthode invoquée était GET ou HEAD. Un client ne peut automatiquement rediriger une requête plus de 5 fois. Il est supposé, si cela arrive, que la redirection s'effectue selon une boucle infinie.

300 Choix multiples

Ce code n'est pas utilisé directement par les applications HTTP/1.0 mais est défini pour servir de réponse par défaut à des codes 3xx non reconnus.

Il signifie en principe que la ressource visée est disponible en un ou plusieurs autres points du réseau. Sauf dans le cas d'une requête de type HEAD, la réponse doit contenir une entité dans laquelle sont listées les caractéristiques et localisations de la ressource demandée, à charge de l'utilisateur de choisir laquelle est la plus appropriée. Si le serveur affiche une préférence de choix, il doit en inscrire l'URL dans un champ Location; Certains clients utiliseront ce champ pour activer une redirection automatique.

301 Changement d'adresse définitive

La ressource demandée est désormais accessible via une autre URL, toute nouvelle requête lui étant appliquée devant utiliser cette nouvelle URL. Les clients implémentant

une fonction de redirection automatique l'activeront et réémettront une requête avec l'URI correcte, lorsque c'est possible.

La nouvelle URL sera indiquée dans le champ d'en-tête Location de la réponse. Sauf dans le cas d'une requête de type HEAD, le corps d'entité de la réponse contiendra une note succincte avec un hyperlien vers la nouvelle URL.

Si le code d'état 301 est reçu en réponse à une requête POST, le client ne pourra rediriger automatiquement la requête sans en avoir demandé confirmation à l'utilisateur. En effet, il n'est pas dit que les conditions ayant été à l'origine de la requête n'aient pas changé.

Note: Certains clients, lorsqu'ils redirigent une requête POST, transforment par erreur cette requête en une requête GET après réception d'un code 301.

302 Changement d'adresse temporaire

La ressource demandée est actuellement et temporairement accessible via une autre URL. La redirection n'étant pas définitive, le client continuera d'utiliser l'URI-visée originale.

La nouvelle URL temporaire doit être spécifiée en réponse dans un champ Location. Sauf dans le cas d'une requête de type HEAD, le corps d'entité de la réponse contiendra une note succincte avec un hyperlien vers la nouvelle URI.

Si le code d'état 302 est reçu en réponse à une requête POST, le client ne pourra rediriger automatiquement la requête sans en avoir demandé confirmation à l'utilisateur. En effet, il n'est pas dit que les conditions ayant été à l'origine de la requête n'aient pas changé.

Note : Certains clients, lorsqu'ils redirigent une requête POST, transforment par erreur cette requête en une requête GET après réception d'un code 302.

304 Non modifié

Ce message est émis en retour d'une requête GET conditionnelle, lorsque l'accès à la ressource est permis, mais que celle-ci n'a pas été modifiée depuis la date et l'heure précisée dans le champ If-Modified-Since de la requête. Le serveur n'émet aucune entité liée à ce message. L'en-tête contiendra des informations à destination du gestionnaire de cache, ou ayant été modifiées sans que cela n'intervienne sur la date de dernière modification de la ressource elle-même. On y trouvera par exemple les champs: Date, Server, et Expires. Un système de cache recevant ce message devra remettre à jour les entités qu'il gère.

9.4 Erreur client 4xx

La classe 4xx de codes d'état est définie pour répondre au cas où il semble que le client ait commis une erreur. Si le client n'a pas encore achevé la transmission de sa requête lorsqu'il reçoit le code 4xx, alors il doit cesser toute transmission. Excepté lorsque ce code répond à une requête de type HEAD, le serveur pourra y inclure une entité explicitant la nature de l'erreur, et indiquant s'il s'agit d'une condition d'erreur temporaire ou permanente. Ces codes sont valides pour tous les types de requête.

Note : Si le client émet des données, les implémentations de TCP devront s'assurer que le client a bien émis tous les accusés de réception des paquets transportant la réponse avant de couper la connexion d'entrée. Si le client continue d'envoyer des données alors que le serveur a fermé la liaison, le TCP serveur émettra en retour un paquet RST (reset), qui risque de stimuler un effacement prématuré de toutes les données reçues dans les tampons interne du TCP client (y compris les données concernant la réponse) avant que celles-ci n'aient pu être transmises à l'application HTTP.

400 Requête incorrecte

La requête n'a pu être reconnue par le serveur pour cause d'une syntaxe incorrecte. Le client n'est pas sensé émettre de nouveau cette requête sans l'avoir corrigée au préalable.

401 Non autorisé

La requête demandée nécessite une authentification de la part de l'utilisateur. La réponse doit inclure un champ d'en-tête WWW-Authenticate (Section 10.16) indiquant la demande d'authentification de la ressource. Le client est sensé répéter la demande en spécifiant un champ d'en-tête d'authentification correct (Section 10.2). Si la requête comportait déjà des données d'authentification, la réponse 401 indique les droits sont actuellement insuffisants pour cette authentification. Si la réponse 401 contient la même demande que la réponse précédente, et l'utilisateur a déjà suivi le processus d'identification au moins une fois, l'entité présentée dans la réponse doit être présentée à l'utilisateur, dans la mesure où les informations qu'elle contient peuvent être de nature à expliquer la faute. L'authentification des accès HTTP est décrite en détail en Section 11.

403 Interdit

Le serveur a compris la requête, mais refuse de la satisfaire.

Une démarche d'authentification n'y fera rien et cette requête ne doit pas être renouvelée. Si la méthode invoquée est différente de HEAD et le serveur souhaite rendre public la raison pour laquelle il refuse le traitement, il le fera dans l'entité liée à cette réponse. Ce code d'état est souvent utilisé lorsque le serveur ne souhaite pas s'étendre sur les raisons pour lesquelles il refuse un accès, ou parce que c'est la seule réponse qui convienne.

404 Non trouvé

Le serveur n'a trouvé aucune ressource correspondant à l'URI-visée. Aucune indication n'est donnée pour savoir si l'erreur est temporaire ou permanente. Si le serveur ne désire pas donner les raisons de l'échec dans ce cas, il peut utiliser le message de code 403 à la place de celui-ci.

9.5 Erreur serveur 5xx

Les réponses de code d'état débutant par un "5" indiquent une situation dans laquelle le serveur sait qu'il est la cause de l'erreur, ou est incapable de fournir le service demandé, bien que la requête ait été correctement formulée. Si le client reçoit cette réponse alors qu'il n'a pas encore terminé d'envoyer des données, il doit cesser immédiatement toute émission vers le serveur. Excepté lorsque la requête invoquée est de type HEAD, le serveur peut inclure une entité décrivant les causes de l'erreur, et s'il

s'agit d'une condition permanente ou temporaire. Ces réponses s'appliquent quelque soit la requête, et ne nécessitent pas de champs d'en-tête particuliers.

500 Erreur serveur interne

Le serveur a été en présence d'un événement inattendu, qui l'a empêché de traiter la requête correctement.

501 Non implémenté

Le serveur ne supporte pas les fonctionnalités requises pour satisfaire la requête. Ceci est typique du cas où malgré une syntaxe conforme, le serveur ne reconnaît pas la méthode invoquée, et ne peut l'appliquer sur aucune ressource.

502 Erreur de routeur

Ce serveur, agissant en tant que proxy ou routeur, a reçu une réponse invalide de la part du serveur amont contacté pour satisfaire la requête.

503 Service indisponible

Les serveur ne peut momentanément traiter la requête, pour cause de surcharge ou de maintenance. Ceci implique une condition de faute temporaire, qui peut disparaître après un certain laps de temps .

Note : L'existence de ce code d'état 503 n'implique pas que le serveur l'utilise dès qu'il est surchargé. Certains serveurs dans cette situation refuseront tout bonnement la connexion.

10. Définition des champs d'en-tête

Cette section décrit la syntaxe et la sémantique des champs d'en-tête essentiels utilisés dans le cadre d'HTTP/1.0. Pour les champs généraux et d'entité, l'émetteur et le récepteur peuvent tour à tour désigner le client ou le serveur, suivant celui qui est à l'origine de la transaction.

10.1 Allow

Le champ Allow (entité) liste l'ensemble des méthodes supportées par la ressource désignée par l'URI-visée. La fonction de ce champ est simplement d'informer le récepteur sur les méthodes qu'il peut utiliser sur la ressource. Ce champ n'est pas autorisé dans un requête de type POST, et doit être ignoré s'il apparaît dans ce dernier cas.

```
Allow                = "Allow" ":" 1#méthode
```

Exemple:

```
                              Allow: GET, HEAD
```

Ce champ ne pourra prévenir le client de tenter d'appliquer d'autres méthodes que celles acceptées par la ressource. Il n'est là qu'à titre informatif. Les méthodes acceptées sont définies par le serveur origine pour chaque requête reçue.

Un proxy ne doit pas modifier le champ d'en-tête Allow même s'il ne connaît pas toutes les méthodes spécifiées, car l'utilisateur peut avoir à sa disposition d'autres moyens de communiquer avec le serveur origine.

Le champ d'en-tête Allow n'indique pas si les méthodes indiquées sont implémentées par le serveur (seulement si elles sont à priori acceptables par la ressource).

10.2 Authorization

Un utilisateur désireux de s'identifier auprès d'un serveur—en général, mais pas nécessairement, après réception d'une réponse 401—peut le faire en incluant un champ d'en-tête Authorization dans sa nouvelle requête. Ce champ contiendra les données d'autentification de l'utilisateur pour la ressource considérée.

```
Authorization        = "Authorization" ":" données_identification
```

L'authentification de connexions HTTP est décrite en Section 11. Si la requête est authentifiée, et un domaine d'accès attribué, les mêmes paramètres d'authentification seront reconnus pour toute autre ressource appartenant à ce domaine.

Les réponses à une requête contenant des données d'identification ne peut être enregistrée en cache.

10.3 Content-Encoding

Le champ Content-Encoding (entité) est utilisé pour compléter l'information de type de média. Lorsqu'il est présent, il indique sous quel codage la ressource est enregistrée, et par voie de conséquence, quel traitement doit être opéré sur l'entité pour pouvoir exploiter celle ci sous son type de média original, défini dans le champ d'en-tête Content-Type. Le champ Content-Encoding a été à l'origine intauré pour permettre la mise à disposition de ressource sous forme compressées, de sorte qu'il soit possible d'en connaître le type de média d'origine.

```
Content-Encoding      = "Content-Encoding" ":" type_codage
```

Les valeurs actuellement reconnues sont décrites en Section 3.5. Exemple:

```
Content-Encoding: x-gzip
```

Le type d'encodage est une caractéristique de la ressource identifiée par l'URI-visée. Typiquement, la ressource est enregistrée encodée, et le décodage ne se fera qu'à l'utilisation finale de celle-ci.

10.4 Content-Length

Le champ d'en-tête Content-Length (entité) indique la taille du corps d'entité, sous la forme d'un nombre d'octets exprimé en décimal, envoyé au récepteur. Dans le cas d'une requête de type HEAD, il renvoie la taille du corps d'entité que le serveur aurait renvoyé si la ressource avait fait l'objet d'une requête GET.

```
Content-Length        = "Content-Length" ":" 1*DIGIT
```

Exemple:

```
Content-Length: 3495
```

Les applications utiliseront ce champ pour transmettre la taille de l'entité jointe, indépendamment du type de média de l'entité. Un champ Content-Length avec une valeur valide est obligatoire dans toute requête HTTP/1.0 contenant un corps d'entité.

Toute valeur numérique supérieure ou égale à zéro est valide. La section 7.2.2 decrit comment évaluer la taille d'un corps d'entité de réponse, si ce champ est omis.

Note: La signification de ce champ est légèrement différente de celle de son équivalent MIME, lequel est un champ optionnel défini à l'intérieur du type "message/external-body". Pour HTTP, il doit être précisé même si la longueur du corps d'entité peut être connu avant la transmission.

10.5 Content-Type

Le champ d'en-tête Content-Type (entité) indique le type de média envoyé au récepteur dans le corps d'entité, ou, si la méthode invoquée est HEAD, le type de média du corps d'entité qui aurait été envoyé si la ressource avait fait l'objet d'une requête de type GET.

```
Content-Type          = "Content-Type" ":" type_de_média
```

Les types de média sont définis en Section 3.6. Exemple:

```
Content-Type: text/html
```

Une présentation d'autres méthodes pour identifier le type de média est donnée en Section 7.2.1.

10.6 Date

Le champ d'en-tête Date (général) donne la date et l'heure à laquelle le message a été "rédigé", et utilise la sémantique de dates de la RFC 822. La valeur de ce champ est une date dans un des formats HTTP décrits à la Section 3.3.

```
Date                  = "Date" ":" date_HTTP
```

Exemple:

```
Date: Tue, 15 Nov 1994 08:12:31 GMT
```

Si un message est reçu en direct du client (pour les requêtes) ou du serveur origine (pour les réponses), on peut affirmer qu'il s'agit aussi de la date d'arrivée au récepteur. En outre, dans la mesure où la date — délivrée par l'origine — est un paramètre important pour tout ce qui touche à la gestion des caches, il est vivement conseillé que les serveurs origine renseignent systématiquement ce champ à la constitution de tout message. Les clients peuvent se limiter à l'envoi d'une date lorsque la requête contient un corps d'entité, comme dans le cas d'une requête POST, et encore cette pratique n'est pas obligatoire. Un message reçu et non daté se verra assigner une date par le récepteur si ce message doit être enregistré en cache ou rerouté via un protocole qui exige une Date.

En théorie, la date doit représenter l'instant juste avant l'émission. En pratique, c'est à la constitution du message que la date est inscrite.

10.7 Expires

Le champ d'en-tête Expires (entité) indique la date et l'heure à partir de laquelle le message doit être considéré comme obsolète. Ceci permet de suggérer la notion de "validité temporaire" de l'information. Les applications ne doivent pas enregistrer ces entités dans leur cache après la date spécifiée. Le présence d'un champ Expires ne signifie pas que la ressource originale n'a pas changé ou n'existe plus à cette date, avant, ou après cette date. Cependant, tout serveur sachant, ou pouvant supposer que cette ressource aura changé à une certaine date devra de préférence inclure un champ Expires

avec cette date. Le format de la date mentionnée est une date absolue telle que définie à la Section 3.3.

```
Expires                = "Expires" ":" date-HTTP
```

Exemple:

```
Expires: Thu, 01 Dec 1994 16:00:00 GMT
```

Si la date indiquée dans ce champ est égale ou antérieure à la date mentionnée dans le champ date du même en-tête, le serveur ou routeur, ou application ne devra pas enregistrer cette ressource dans son cache. Si la ressource est dynamique par nature, comme c'est le cas pour des résultats de processus générateurs de données, les entités produites se verront attribuées un champ Expires renseigné d'une façon correspondante à ce "dynamisme".

Le champ Expires ne peut pas être utilisé pour forcer un client à rafraîchir ou recharger une ressource. Sa sémantique ne concerne que les mécanismes de gestion du cache, lesquels n'utilisent ce paramètre que pour savoir si le serveur peut encore utiliser l'instance cachée lorsqu'une requête destinée à ce document est à nouveau présentée.

Les clients HTTP disposent souvent d'un mécanisme d'historique, implémenté sous forme d'un bouton "Back" et/ou d'une liste de "documents précédents", utilisés pour recharger un document précédemment chargé pendant la session courante. Par défaut, le champ Expires n'influe pas sur ces mécanismes. Si l'entité est stockée par un tel mécanisme, elle pourra être rechargée même si sa date d'expiration est passée, sauf si l'utilisateur a explicitement configuré le client pour régénérer les documents ayant expiré.

Note : Il est encouragé que les applications fassent preuve d'une certaine tolérance quant à des implémentations erronées ou incomplètes du champ Expires. Une valeur zéro (0) ou une date présentée sous un mauvais format signifient "expiration immédiate". Bien que ces valeurs ne soient pas valides dans le cadre d'HTTP/1.0, une implémentation tolérante est toujours souhaitable.

10.8 From

Le champ d'en-tête From (requête), si présent, devra contenir l'adresse e-mail Internet de l'utilisateur "humain" contrôlant le client émetteur de la requête. Cette adresse doit être codée sous une forme utilisable par les machines, telle que définie dans la RFC 822 [7] (et RFC 1123 [6]):

```
From                   = "From" ":" adresse_mail
```

Exemple:

```
From: webmaster@w3.org
```

Ce champ est exploité à des fins de statistiques, et pour identifier la source de requêtes non valides ou non autorisées. Il ne doit cependant pas être utilisé dans un contexte de sécurisation d'accès. Il doit être interprété comme l'identification de la personne ayant formulé la requête, et qui accepte la responsabilité de la méthode

employée. En particulier, les robots devront renseigner ce champ afin que son responsable d'exploitation puisse être contacté en cas de dysfonctionnement à l'autre extrémité.

L'adresse e-mail Internet dans ce champ doit être séparé de la mention du nom de l'ordinateur d'où a été émis la requête. Par exemple, lorsque la requête a transité par l'intermédiaire d'un proxy, l'adresse de la source originale doit y être mentionnée.

Note : Le client ne peut donner de valeur pour ce champ Form sans l'autorisation de l'utilisateur, car cette information rentre dans le cadre de la protection des données privées et des droits individuels. Il est fortement recommandé que le client laisse le choix à l'utilisateur de pouvoir activer, désactiver, et modifier la valeur émise dans ce champ avant émission de toute requête.

10.9 If-Modified-Since

Le champ d'en-tête If-Modified-Since (requête) est utilisé lors d'une requête de type GET pour en émettre une forme conditionnelle: Si la ressource visée n'a pas été modifiée depuis la date mentionnée dans ce champ, la copie de la ressource ne sera pas renvoyée par le serveur; En lieu et place, une réponse de type 304 (non modifié) sera renvoyée sans aucun corps d'entité associé.

```
If-Modified-Since    = "If-Modified-Since" ":" date_HTTP
```

Exemple:

```
If-Modified-Since: Sat, 29 Oct 1994 19:43:31 GMT
```

Une méthode GET conditionnelle requiert la transmission d'une ressource uniquement si celle-ci a été modifiée après la date indiquée dans le champ If-Modified-Since. L'algorithme utilisé pour déterminer cette condition prend en compte les cas suivants:

a) Si la résolution de la requête conduit à une réponse autre que 200 (OK), ou si la date passée par le champ If-Modified-Since n'est pas valide, la réponse sera alors donnée comme si la méthode invoquée était un GET standard. Une date postérieure à la date courante du serveur est non valide.
b) Si la ressource a été modifiée après la date If-Modified-Since, la réponse sera donnée comme dans le cas d'une méthode GET standard.
c) Si la ressource n'a pas été modifiée depuis la date If-Modified-Since, et cette date est valide, le serveur renverra une réponse 304 (non modifié).

Le but de cette implémentation est de permettre une remise à jour efficace des informations en cache, en réduisant au maximum les transactions réseau.

10.10 Last-Modified

Le champ d'en-tête Last-Modified (entité) indique la date et l'heure à laquelle le serveur pense que la ressource visée a été modifiée pour la dernière fois. La sémantique exacte de ce champ dépend fortement de la manière dont le récepteur va le comprendre: Si le récepteur dispose d'une copie de cette ressource d'une date inférieure à celle

mentionnée dans le champ Last-Modified, cette copie devra être considérée comme obsolète.

```
Last-Modified         = "Last-Modified" ":" date_HTTP
```

Exemple:

```
Last-Modified: Tue, 15 Nov 1994 12:45:26 GMT
```

La signification exacte de cette information dépend de l'implémentation de l'émetteur et de la ressource à laquelle elle s'applique. Pour des fichiers, il s'agira de la date de dernière modification donnée par le gestionnaire de fichier du système d'exploitation. Pour des entités incluant des parties générées dynamiquement, cette date peut être plus récente que les dates de dernière modification de chacune de ses parties. Pour des routeurs vers des bases de données, il pourra s'agir de la dernière étiquette de date associée à l'enregistrement. Pour des objets virtuels, il pourra s'agir de la dernière date à laquelle l'état interne de l'objet a changé.

Un serveur origine ne doit pas envoyer de date Last-Modified postérieure à la date d'émission du message (date locale du serveur). Dans une telle éventualité, où la date à inscrire pointerait vers un instant futur, c'est la date d'émission du message qui doit être inscrite.

10.11 Location

Le champ d'en-tête Location (réponse) renvoie la localisation exacte de la ressource identifiée par l'URI-visée de la requête. Pour des réponses 3xx, ce champ indique l'URL "de préférence" donnée par le serveur pour la fonction de redirection automatique. Une seule URI absolue peut être mentionnée à la fois.

```
Location              = "Location" ":" URI_absolue
```

Exemple:

```
Location: http://www.w3.org/hypertext/WWW/NewLocation.html
```

10.12 Pragma

Le champ d'en-tête Pragma (générale) est utilisé pour transmettre des directives dépendantes de l'implémentation à tous les agents de la chaîne de requête/réponse. Toutes les directives Pragma spécifient un comportement particulier optionnel de la part des agents vis à vis du protocole; cependant, certains systèmes ne pourront répondre à toutes les directives.

```
Pragma                = "Pragma" ":" 1#directive_pragma

directive_pragma      = "no-cache" | pragma
pragma                = nom_de_pragma [ "=" mot ]
```

Lorque la directive "no-cache" est présente dans un message de requête, les applications doivent répercuter la requête jusqu'au serveur origine, même si elles disposent d'une copie de la ressource en cache. Ceci permet au client de forcer la récupération d'une "première copie" d'une ressource. Ceci permet de plus de demander au client une remise à jour de copies cachées, dans le cas où ces copies se sont avérées défectueuses, ou obsolètes.

Les directives Pragma doivent être réémises par les routeurs ou proxies, quelle que soit leur signification à l'égard des applications, dans la mesure où ces directives concernent toutes les applications de la chaîne de requête/réponse. Il n'est pas possible de définir un pragma ne concernant qu'un intermédiaire donné dans la chaîne; Cependant, toute directive non reconnue par l'un des récepteurs sera ignorée.

10.13 Referer

Le champ d'en-tête Referer (requête) permet au client d'indiquer, à l'usage du serveur, l'adresse (URI) de la ressource dans laquelle l'URI-visée a été trouvée. Ceci permet au serveur de générer et entretenir des listes de "rétro-liens" destinées à renseigner les clients futurs sur des "sites intéressants", ou à but d'archivage et d'analyse, optimisation de cache, etc. Il permet aussi l'analyse de liens obsolètes ou de type incorrect à but de maintenance. Le champ Referer ne doit pas être inscrit si la source de l'URI-visée provient d'une entité n'ayant pas d'URI propre, comme l'entrée de l'adresse via un clavier.

```
Referer              = "Referer" ":" ( URI_absolue | URI_relative )
```

Exemple:

```
    Referer: http://www.w3.org/hypertext/DataSources/Overview.html
```

Si une URI partielle est donnée, elle se référera relativement à l'URI-visée. L'URI donnée en référence ne doit pas inclure de fragment.

Note : Dans la mesure où la référence d'un lien est une information qui peut avoir un caractère privé, ou être amenée à révéler une information privée, il est recommandé de laisser le choix à l'utilisateur final de renseigner ou non ce champ. Par exemple, un navigateur pourra disposer d'un commutateur qui permettra une navigation ouverte ou anonyme, qui simultanément peut activer ou désactiver l'émission des champs Referer et Form.

10.14 Server

Le champ d'en-tête Server (réponse) contient des informations sur le logiciel utilisé par le serveur origine pour traiter la requête. Ce champ peut contenir plusieurs noms de produits différents (Section 3.7) ainsi que des commentaires identifiant le serveur et des "sous-composants" des applications. Par convention, les noms sont listés par ordre d'importance, eu égard à l'identification du produit utilisé.

```
Server               = "Server" ":" 1*( produit | commentaire )
```

Exemple:

```
                    Server: CERN/3.0 libwww/2.17
```

Si la réponse est transmise via un proxy, ce dernier ne doit pas ajouter ses propres commentaires à la liste.

Note : La révélation du nom et de la version du logiciel serveur peut présenter le risque de permettre des attaques extérieures contre telle ou telle application dont les "portes dérobées" sont connues. Les développeurs de serveurs auront tout intérêt à laisser l'émission de cette information optionnelle.

10.15 User-Agent

Le champ d'en-tête User-Agent (requête) contient des informations sur l'utilisateur émetteur de la requête. Cette information est utilisée à des fins statistiques, pour tracer des violations de protocole, et à des fins de reconnaissance automatique d'utilisateur permettant de formater la réponse de la manière la plus adaptée. L'usage de ce champ, bien que non indispensable, est cependant conseillé. Le champ User-Agent peut mentionner plusieurs noms de produits (Section 3.7) ainsi que des commentaires identifiant le programme ou des sous-composants de ce programme dans la mesure où ceux-ci peuvent être significatifs pour le serveur. Par convention, ces informations sont listées par ordre d'importance.

```
User-Agent          = "User-Agent" ":" 1*( produit | commentaires )
```

Exemple:

```
User-Agent: CERN-LineMode/2.15 libwww/2.17b3
```

Note : Certains proxies ajoutent leur propre information à ce champ. Ceci est déconseillé, dans la mesure où le contenu final de ce champ peut alors devenir ambigu, voire confus.

10.16 WWW-Authenticate

Le champ d'en-tête WWW-Authenticate (réponse) doit être inclus dans des réponses de type 401 (non autorisé). La valeur du champ consiste en un ou plusieurs "modèles" d'identification pour l'URI-visée.

```
WWW-Authenticate     = "WWW-Authenticate" ":" 1#modèle
```

Le processus d'identification et d'authentification d'accès HTTP est décrit en Section 11. Les applications utilisatrices doivent interpréter avec circonspection la valeur d'un champ WWW-Authenticate lorsqu'il présente plus d'un modèle d'authentification, ou si plusieurs champs WWW-Authenticate apparaissent dans l'en-tête, le contenu d'un modèle pouvant lui même apparaître comme une liste de paramètres séparés par une virgule.

11. Authentification d'accès sous HTTP

HTTP propose un mécanisme simple de "modèle-accréditif" pour permettre à un serveur de fournir un modèle d'identification et d'authentification à un client, et à un client de s'authentifier pour une requête particulière. Ce mécanisme utilise un système de noms extensible et indépendant de la casse, dans le but d'identifier le schéma d'authentification, suivie d'une liste de paires "attribut/valeur" séparées par des virgules, destinées à transmettre les paramètres nécessaires pour achever l'authentification.

```
Scheme-auth          = nom_de_scheme

param-auth           = nom_param "=" chaîne entre guillemets
```

Le message de réponse 401 (non autorisé) est utilisé par un serveur origine pour soumettre le "modèle" d'authentification à un client. Cette réponse doit comporter un champ d'en-tête WWW-Authenticate proposant au moins un modèle valide pour la ressource à atteindre.

```
Modèle               = scheme-auth 1*SP domaine_valide *( "," param-
                     auth )

domaine_valide       = "realm" "=" espace_domaine
espace-domaine       = chaîne entre guillemets
```

L'attribut de domaine (indépendant de la casse) est requis dans tous les schèmes d'authentification qui soumettent un modèle. L'espace de domaine (indépendant de la casse), combiné à l'URL canonique du serveur origine, définit l'espace protégé. Cette technique permet de partitionner un serveur protégé en plusieurs sous ensembles, protégés individuellement, chacun avec leur propre modèle et leurs propres paramètres d'autorisation liées ou non à une base de donnée d'authentification. Le domaine accessible est exprimé sous forme de chaîne de caractères, en général donnée par le serveur origine, avec éventuellement une sémantique supplémentaire dépendant du modèle d'authentification.

Un utilisateur désireux de s'authentifier auprès d'un serveur – en général, mais pas nécessairement, après avoir reçu une réponse 401—peut le faire en spécifiant un champ Authorization dans l'en-tête de sa requête. La valeur dans le champ Authorization consiste contient l'accréditif nécessaire à l'accès au domaine autorisé pour la ressource visée.

```
accréditif  = accréditif_de_base | ( modèle_authentification#paramètres
                      )
```

Le domaine accessible par un utilisateur utilisant cet accréditif est déterminé par les données de protection du serveur. Si une requête précédente à abouti à une autorisation d'accès, le même accréditif donnera accès au même domaine accessible durant un temps déterminé par le modèle d'authentification, ses paramètres, et/ou les préférences utilisateur. Sauf mention explicite dans le modèle, un espace protégé ne peut sortir du cadre du serveur qui le gère.

Si le serveur refuse l'accès à une requête, il délivrera en retour une réponse 403 (accès interdit).

Le protocole HTTP n'exclut pas l'utilisation de schémas de protection additionnels, venant compléter ou remplacer le paradigme "modèle-accréditif". D'autres mécanismes, tels que le cryptage au niveau "transport" ou l'encapsulation de messages, peuvent être utilisés avec des champs d'en-tête additionnels véhiculant les informations d'authentification. Ces mécanismes ne sont toutefois pas décrits dans cette spécification.

Les proxies doivent être complètement transparents vis à vis du mécanisme d'authentification. C'est à dire qu'ils doivent retransmettre les champs WWW-Authenticate et Authorization tels que, et ne doivent pas enregistrer une réponse à un message contenant le champ Authorization dans leur cache. HTTP/1.0 ne fournit aucun moyen à un client de s'identifier vis à vis d'un proxy.

11.1 Modèle d'authentification de base

Le modèle dit "de base" demande à un utilisateur de s'authentifier par un user-ID et un mot de passe pour chaque "domaine d'accès". La donnée d'authentification doit représenter comme une chaîne non visible, pouvant seulement être comparée à d'autres accréditifs de référence par le serveur concerné. Le serveur n'autorisera l'accès que si et l'user-ID, et le mot de passe correspondent à un schème d'authentification défini pour le domaine auquel appartient l'URI-visée. Il n'y a pas de paramètres optionnels pour ce modèle.

Sur toute réception d'une requête non autorisée visant une URI dans l'espace protégé, le serveur doit renvoyer une demande d'identification de forme:

```
WWW-Authenticate: Basic realm="WallyWorld"
```

dans laquelle "WallyWorld" est le nom symbolique de l'espace contenant l'URI-visée, attribué par le serveur.

Pour obtenir l'autorisation d'accès, le client enverra l'user-ID et le mot de passe, séparés par un "deux-points" (":"), le tout encodé en base64 [5].

```
Accréditif_de_base   = "Basic" SP cookie-basique

cookie-basique       = <"userID-mot_de_passe" encodé base64 [5], limité
                     à 76 char/line>

userID-mot_de_passe  = [ nom_ID ] ":" *TEXT
```

Si le client voulait utiliser l'user-ID "Aladdin" et le mot de passe "open sesame", il spécifierait le champ ainsi:

```
Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==
```

Le modèle de base ci-défini est une méthode non sécurisée de filtrage d'accès à des ressources données sur un serveur HTTP. Il suppose en effet que la connexion entre l'utilisateur et le serveur est un lien digne de confiance. Ceci n'est pas le cas sur des réseaux ouverts, et ce modèle doit être utilisé en connaissance de cause. Malgré cette faiblesse, les clients devront implémenter ce modèle afin de pouvoir communiquer avec les serveurs qui l'utilisent.

12. Sécurité

Cette section est une information à destination des développeurs d'applications, hébergeurs de données, et des utilisateurs, concernant les limites de sécurisation atteintes par HTTP/1.0 tel que décrit dans ce document. La discussion suivante ne prétend pas apporter de solution opérationnelle aux défauts qui y sont révélés, bien que quelques suggestions y soient faites pour réduire le risque informatique.

12.1 Authentification des clients

Comme le mentionne la Section 11.1, le modèle d'authentification "de base" n'est pas une méthode infaillible pour prévenir des accès non autorisés. Il ne peut non plus empêcher la transmission du corps d'entité "en clair" sur le réseau physique utilisé comme porteuse. HTTP/1.0 n'interdit aucunement l'emploi d'autres modèles, ou autres mécanismes de protection afin d'accroître la sécurité de transmission.

12.2 Méthodes sûres

Les éditeurs de logiciels clients doivent garder à l'esprit que leur application représente l'utilisateur dans ses interactions avec Internet, et doivent de ce fait s'assurer que l'utilisateur sera averti de toute action que l'application peut entamer, et susceptible d'avoir un résultat inattendu, tant pour eux que pour les tiers.

En particulier, il a été établi par convention que les méthodes GET et HEAD ne peuvent signifier autre chose qu'une demande de récupération de ressource. Ces méthodes peuvent être considérées comme "sûres". Ceci laisse la possibilité aux clients d'implémenter d'autres méthodes (ex. POST) sous une forme spécifique, mais toujours à condition que l'utilisateur puisse être averti qu'une action peu sûre ou potentiellement non conforme est requise.

Naturellement, il n'est pas possible de garantir que le serveur n'aura pas "d'effets de bord" lorsqu'il traite une requête GET; en fait, certaines ressources dynamiques tirent avantage de cette possibilité. La seule distinction de taille est que l'utilisateur qui émet une requête sûre de type GET n'a pas "dans l'intention" de provoquer ces effets de bord. Il ne peut donc en être tenu pour responsable.

12.3 Abus de l'information Server Log Information

Un serveur a toute possibilité de stocker des données personnelles contenues dans les requêtes telles qu'adresses et modèles d'accès ou en déduire et archiver des conclusions

personnelles sur les goûts, les tendances ou les centres d'intérêt des utilisateurs. Ces informations sont par nature typiquement confidentielles et peuvent tomber sous le coup de la loi dans certains pays (NdT: typiquement dans le cadre de la loi "informatique et libertés", dans la mesure où un archivage systématique et une interprétation "sociologique" nominative sont réalisés). Les personnes utilisant le protocole HTTP pour diffuser des données ont la responsabilité de s'assurer que toute information reçue pouvant identifier et classifier les visiteurs ne puissent être communiquée sans l'accord de ces derniers.

12.4 Transfert d'information sensible

Comme tout protocole de transmission de donnée de base, HTTP ne peut contrôler la nature des informations transmises. Il n'est pas non plus possible à priori de déterminer la "sensibilité" de certaines portions de données transmises dans le cadre d'une requête quelle qu'elle soit. C'est pourquoi les applications devront prendre en charge la plus grande part du contrôle de ces informations en lieu et place du diffuseur. Trois champs d'en-tête sont particulièrement concernés par cette notion: Server, Referer et From.

Révéler la version exacte du logiciel serveur peut rendre celui-ci vulnérable aux attaques si le logiciel est réputé avoir des faiblesses en termes de sécurité. Le renseignement du champ Server doit de ce fait rester optionnel, et ne doit pas forcément être systématique.

Le champ Referer autorise l'étude d'un certain nombre d'informations à propos de la requête et permet de "remonter" une chaîne de liens. Bien que cette fonction soit très utile, il est possible néanmoins d'en abuser si les informations sur l'utilisateur ne sont pas dissociées de celles contenues dans le Referer. Même lorsque ces informations "personnelles" sont enlevées, le champ Referer peut parfois indiquer l'URI d'un document privé dont la publication n'est pas souhaitable.

L'information émise dans le champ From peut entrer en conflit avec la défense des droits privés, ou peut diminuer le degré de sécurité du serveur dans lequel la ressource est émise de ce fait, toute implémentation devra laisser la possibilité à l'utilisateur d'émettre, ne pas émettre, ou modifier les données dans ce champ. L'utilisateur devra être en mesure de configurer une valeur "par défaut" ou une "préférence utilisateur" pour les données dans ce champ.

Nous suggérons, mais n'imposons pas, qu'un interrupteur graphique soit implémenté sur l'interface utilisateur pour permettre ou interdire l'envoi des informations From et Referer.

12.5 Attaques sur les Fichiers et Répertoires

Les implémentations des serveurs origine HTTP devront veiller à restreindre la transmission des documents demandés par une requête HTTP aux seuls documents autorisés par l'administration système. Si un serveur HTTP traduit les URI HTTP directement en appels système de fichiers, le serveur devra veiller à ne pas livrer de documents destinés à autre chose qu'un client HTTP. Par exemple, Unix, Microsoft Windows, et d'autres systèmes d'exploitation utilisent ".." comme accès symbolique au répertoire de niveau supérieur. Sur un tel système, un serveur HTTP doit refuser une telle construction dans l'URI-visée, interdisant ainsi l'accès potentiel à des données qui ne sont pas sensées être diffusées par un serveur HTTP. De même, tous les fichiers à usage interne au serveur (fichiers de contrôle d'accès, de configuration, et codes script) doivent

être protégés contre toute diffusion, dans la mesure où ils peuvent contenir des informations essentielles et confidentielles. L'expérience a montré que des imperfections mineures du logiciel serveur ont pu conduire à un réel risque en terme de sécurité.

13. Crédits

Cette spécification utilise abondamment le format BNF étendu et les constructions génériques définies par David H. Crocker pour la RFC 822 [7]. De plus, elle réutilise de nombreuses définitions établies par Nathaniel Borenstein et Ned Freed pour la définition des MIME [5]. Nous espérons que cette utilisation permettra de lever les confusions entre la sémantique HTTP/1.0 et les formats de message du service "courrier électronique".

Le HTTP protocole a évolué considérablement ces quatre dernières années. Il a bénéficié de la réflexion d'une large et dynamique communauté de développeurs – les nombreuses personnes qui ont participé aux groupes de travail du WWW – laquelle a été largement responsable du développement fulgurant d'HTTP et du World-Wide Web en général. Nous décernons à Marc Andreessen, Robert Cailliau, Daniel W. Connolly, Bob Denny, Jean-Francois Groff, Phillip M. Hallam-Baker, Håkon W. Lie, Ari Luotonen, Rob McCool, Lou Montulli, Dave Raggett, Tony Sanders, et Marc VanHeyningen une mention d'honneur pour la reconnaissance de leur efforts dans la définition des toutes premières spécifications ayant servi de base à la présente.

Paul Hoffman aura contribué aux sections informatives de ce document et aux Appendices C et D.

Ce document a bénéficié des nombreuses remarques et objections de la grande majorité des participants au HTTP-WG. Outre les personnes déjà mentionnées, les personnes suivantes ont collaboré à l'achèvement de cette spécification:

Gary Adams
Keith Ball
Paul Burchard
Mike Cowlishaw
Michael A. Dolan
Jim Gettys
Koen Holtman
Bob Jernigan
Martijn Koster
Daniel LaLiberte
Albert Lunde
Larry Masinter
Jeffrey Mogul
Bill Perry

Harald Tveit Alvestrand
Brian Behlendorf
Maurizio Codogno
Roman Czyborra
John Franks
Marc Hedlund
Alex Hopmann
Shel Kaphan
Dave Kristol
Paul Leach
John C. Mallery
Mitra
Gavin Nicol
Jeffrey Perry

Owen Rees
David Robinson
Rich Salz
Chuck Shotton
Simon E. Spero
François Yergeau
Jean-Philippe Martin-Flatin
Luigi Rizzo
Marc Salomon
Jim Seidman
Eric W. Sink
Robert S. Thau
Mary Ellen Zurko

14. Bibliographie

[1] Anklesaria, F., McCahill, M., Lindner, P., Johnson, D., Torrey, D., and B. Alberti, "The Internet Gopher Protocol: A distributed document search and retrieval protocol", RFC 1436, University of Minnesota, March 1993.

[2] Berners-Lee, T., "Universal Resource Identifiers in WWW: A Unifying Syntax for the Expression of Names and Addresses of Objects on the Network as used in the World-Wide Web", RFC 1630, CERN, June 1994.

[3] Berners-Lee, T., and D. Connolly, "Hypertext Markup Language - 2.0", RFC 1866, MIT/W3C, November 1995.

[4] Berners-Lee, T., Masinter, L., and M. McCahill, "Uniform Resource Locators (URL)", RFC 1738, CERN, Xerox PARC, University of Minnesota, December 1994.

[5] Borenstein, N., and N. Freed, "MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies", RFC 1521, Bellcore, Innosoft, September 1993.

[6] Braden, R., "Requirements for Internet hosts - application and support", STD 3, RFC 1123, IETF, October 1989.

[7] Crocker, D., "Standard for the Format of ARPA Internet Text Messages", STD 11, RFC 822, UDEL, August 1982.

[8] F. Davis, B. Kahle, H. Morris, J. Salem, T. Shen, R. Wang, J. Sui, and M. Grinbaum. "WAIS Interface Protocol Prototype Functional Specification." (v1.5), Thinking Machines Corporation, April 1990.

[9] Fielding, R., "Relative Uniform Resource Locators", RFC 1808, UC Irvine, June 1995.

[10] Horton, M., and R. Adams, "Standard for interchange of USENET messages", RFC 1036 (Obsoletes RFC 850), AT&T Bell Laboratories, Center for Seismic Studies, December 1987.

[11] Kantor, B., and P. Lapsley, "Network News Transfer Protocol: A Proposed Standard for the Stream-Based Transmission of News", RFC 977, UC San Diego, UC Berkeley, February 1986.

[12] Postel, J., "Simple Mail Transfer Protocol", STD 10, RFC 821, USC/ISI,

August 1982.

[13]
Postel, J., "Media Type Registration Procedure", RFC 1590, USC/ISI, March 1994.

[14]
Postel, J., and J. Reynolds, "File Transfer Protocol (FTP)", STD 9, RFC 959, USC/ISI, October 1985.

[15]
Reynolds, J., and J. Postel, "Assigned Numbers", STD 2, RFC 1700, USC/ISI, October 1994.

[16]
Sollins, K., and L. Masinter, "Functional Requirements for Uniform Resource Names." RFC 1737, MIT/LCS, Xerox Corporation, December 1994.

[17]
US-ASCII. Coded Character Set - 7-Bit American Standard Code for Information Interchange. Standard ANSI X3.4-1986, ANSI, 1986.

[18]
ISO-8859. International Standard -- Information Processing --
8-bit Single-Byte Coded Graphic Character Sets --
Part 1: Latin alphabet No. 1, ISO 8859-1:1987.
Part 2: Latin alphabet No. 2, ISO 8859-2, 1987.
Part 3: Latin alphabet No. 3, ISO 8859-3, 1988.
Part 4: Latin alphabet No. 4, ISO 8859-4, 1988.
Part 5: Latin/Cyrillic alphabet, ISO 8859-5, 1988.
Part 6: Latin/Arabic alphabet, ISO 8859-6, 1987.
Part 7: Latin/Greek alphabet, ISO 8859-7, 1987.
Part 8: Latin/Hebrew alphabet, ISO 8859-8, 1988.
Part 9: Latin alphabet No. 5, ISO 8859-9, 1990.

15. Adresses des auteurs

Tim Berners-Lee
Director, W3 Consortium
MIT Laboratory for Computer Science
545 Technology Square
Cambridge, MA 02139, U.S.A.

Fax: +1 (617) 258 8682
EMail: timbl@w3.org

Roy T. Fielding
Department of Information and Computer Science
University of California
Irvine, CA 92717-3425, U.S.A.

Fax: +1 (714) 824-4056
EMail: fielding@ics.uci.edu

Henrik Frystyk Nielsen
W3 Consortium
MIT Laboratory for Computer Science
545 Technology Square
Cambridge, MA 02139, U.S.A.

Fax: +1 (617) 258 8682
EMail: frystyk@w3.org

Appendices

Ces appendices ont été ajoutés par souci d'information – Ils ne forment pas partie intégrante de la spécification HTTP/1.0.

A. Internet Media Type message/http

Outre la définition du protocole HTTP/1.0, ce document sert à la spécification de l'Internet Media Type "message/http". La spécification suivante est enregistrée auprès de l'IANA [13].

```
Media Type :                    message

Media subtype :                 http

Paramètres obligatoires:        none

Paramètres optionnels:          version, msgtype
```

version: Version du protocole HTTP utilisé pour le message courant (e.g., "1.0"). Si ce paramètre est absent, la version peut être déduite par analyse de la première ligne du corps.

msgtype: Le type de message -- "request" ou "response". Si ce paramètre est absent, la version peut être déduite par analyse de la première ligne du corps.

Considérations d'encodage: seulement "7bits", "8bits", ou "binary" permis

Considérations en termes de sécurité: aucune

B. Applications tolérantes

Bien que ce document donne toutes les spécifications pour la génération de messages HTTP/1.0, toutes les applications ne présenteront pas une implémentation correcte. Nous recommandons de ce fait que les applications opérationnelles puissent tolérer certaines déviation de ce protocole, dans la mesure où celles-ci gardent un caractère univoque.

Les clients devront faire preuve de tolérance dans l'interprétation de la ligne d'état. Les serveurs devront à leur tour se montrer ouverts dans l'interprétation de la ligne de

Requête. En particulier, ils devront accepter tout nombre d'espaces ou de tabulations entre les champs, même lorsqu'un espace simple est requis.

La fin de ligne dans les en-têtes HTTP est marquée par la séquence CRLF. Cependant, nous conseillons aux applications interprétant de telles en-têtes de reconnaître une fin de ligne sur LF simple en ignorant le CR de tête.

C. Relations avec les MIME

HTTP/1.0 utilise de nombreuses syntaxes déjà employées pour le Mail Internet (RFC 822 [7]) et entre autres les Multipurpose Internet Mail Extensions (MIME [5]) permettant aux entités d'être transmises sous une grande quantité de formes tout en respectant le principe d'évolutivité. Cependant, la RFC 1521 concerne le mail, et HTTP implémente certaines fonctions d'une façon différente de la RFC 1521. Ces différences ont été soigneusement contrôlées afin d'optimiser la transmission en mode binaire, pour ouvrir le champ d'application offert par des nouveaux types de média, pour faciliter la comparaison de dates, tout en restant compatibles avec des serveurs et clients HTTP antérieurs.

A cette heure, nous espérons une révision de la RFC 1521. Cette révision pourrait intégré quelques unes des pratiques utilisées par HTTP/1.0 mais ne faisant pas partie de la RFC 1521.

Cet appendice décrit les points spécifiques pour lesquels HTTP et la RFC 1521 diffèrent. Les proxies et routeurs dirigeant des messages HTTP vers un environnement MIME strict devront connaître ces différences et propose une conversion appropriée si nécessaire. A l'inverse Les proxies et routeurs accédant à un environnement HTTP à partir d'un environnement MIME strict, devront de même se charger de la conversion, et donc connaître ces différences.

C.1 Conversion vers la forme canonique

La RFC 1521 impose qu'une entité Internet Mail soit convertie dans sa forme canonique avant d'être transmise, (Appendice G de la RFC 1521 [5]). La Section 3.6.1 de ce document décrit toutes les formes admises du sous-type "texte" transmis sous HTTP.

La RFC 1521 impose que le contenu d'un message dont le Content-Type est "text" représente les fins de ligne par la séquence CRLF et interdise l'usage de CR ou de LF en dehors de ce contexte de fin de ligne. HTTP permet l'usage de séquences CRLF, de CR et LF seuls pour marquer la fin de ligne du texte dans le corps du document envoyé sous HTTP.

Là où c'est possible, un proxy ou un routeur du HTTP vers un environnement strictement RFC 1521 devra traduire toutes les fins de ligne du texte contenu dans ces documents concernés par la Section 3.6.1 dans ce document en forme canonique selon la RFC 1521: une séquence CRLF. Notez, cependant, que cette règle peut se compliquer dans le cas où un champ Content-Encoding est présent et par le fait qu'HTTP permettent l'utilisation de tables de caractères tant que le caractère 13 et 10 continuent à représenter les équivalents de CR et LF (comme c'est le cas pour certaines tables utilisant plusieurs octets par caractère).

C.2 Conversion de formats de dates

HTTP/1.0 utilise un ensemble de formats de date restreint (Section 3.3) pour simplifier l'implémentation des comparaisons de date. Les proxies et routeurs agissant à partir d'autres protocoles devront s'assurer que tout champ d'en-tête de Date dans un message

reste conforme à l'un des formats reconnus par HTTP/1.0, et réécriront les dates si nécessaire.

C.3 Introduction du champ Content-Encoding

La RFC 1521 n'introduit aucun concept équivalent au champ d'en-tête Content-Encoding défini par HTTP/1.0. Comme celui-ci fait fonction de modificateur du type de média, les proxies et routeurs depuis HTTP vers des protocoles compatibles MIME doivent soit changer la valeur indiquée dans le champ Content-Type, soit analyser le corps d'entité avant de faire suivre le message. (Certaines implémentations expérimentales du champ Content-Type pour la messagerie Internet ont utilisé une valeur ";conversions=<content-coding>" pour obtenir une fonction similaire à celle procurée par le champ Content-Encoding. Ce paramètre n'est pas officiel dans la RFC 1521.)

C.4 Pas de champ Content-Transfer-Encoding

HTTP n'exploite pas le champ Content-Transfer-Encoding (CTE) utilisé par la RFC 1521. Les proxies et routeurs depuis un protocole MIME vers un protocole HTTP devront supprimer tout CTE à l'exclusion éventuellement du CTE "identity" ("quoted-printable" ou "base64") avant de faire suivre le message vers un client HTTP client.

Les proxies et routeurs depuis HTTP vers un protocole compatible MIME ont la responsabilité de s'assurer que le message envoyé est au format correct pour un transfert en toute sécurité sous ce protocole, cette sécurité étant naturellement limitée aux limitations propres du protocole utilisé. Un tel proxy ou routeur devra ajouter un Content-Transfer-Encoding approprié, de façon à présenter les données conformément aux attentes de ce protocole.

C.5 Champs d'en-tête HTTP dans des parties de corps Multipart

Sous RFC 1521, la plupart des champs d'en-tête placés dans les sous-parties d'un corps Multipart sont en général ignorés sauf si le nom du champ débute par "Content-". En HTTP/1.0, les sous-parties de corps Multipart peuvent contenir tout champ d'en-tête HTTP dont la signification est pertinente pour cette partie de message.

D. Fonctions supplémentaires

Cet appendice liste quelques éléments de protocole parfois employés dans certaines implémentations HTTP, mais qui ne sont pas considérées comme "légitimes" par la plupart des applications HTTP/1.0. Les développeurs ont un intérêt à connaître ces fonctions, mais ne peuvent être sûrs de leur présence effective, ni de leur implémentation par d'autres applications HTTP/1.0.

D.1 Méthodes de requètes supplémentaires

D.1.1 PUT

La méthode PUT demande à ce que l'entité jointe soit enregistrée par le serveur sous l'URI-visée. Si cette URI pointe vers une ressource déjà existante, l'entité jointe sera considérée comme une nouvelle version de celle jusqu'alors présente sur le serveur origine. Si l'URI-visée pointe sur une ressource inexistante, et à la condition que cette URI puisse être définie en tant que nouvelle ressource du serveur, ce dernier créera une nouvelle ressource sous cette URI.

La différence fondamentale entre les méthodes POST et PUT réside dans la signification donnée à l'URI-visée. Celle-ci, pour une méthode POST désigne la ressource "active" à laquelle l'entité doit être confiée dans le but d'un traitement. Cette ressource peut être un programme, un routeur ou un autre protocole, ou encore une entité acceptant des annotations. Par contre, L'URI précisée dans une méthode PUT nomme l'entité incluse dans la requête – Le client sait parfaitement de quelle URI il s'agit et le serveur n'applique la méthode à aucune autre ressource.

D.1.2 DELETE

La méthode DELETE demande au serveur de supprimer la ressource pointée par l'URI-visée.

D.1.3 LINK

La méthode LINK établit une ou plusieurs liaisons entre la ressource existante définie par l'URI-visée et d'autres ressources.

D.1.4 UNLINK

A contrario, cette méthode supprime des liaisons entre la ressource définie par l'URI-visée, et d'autres ressources qui lui sont liées.

D.2 Définitions d'autres champs d'en-tête

D.2.1 Accept

Le champ d'en-tête Accept (requête) peut être utilisé pour définir une liste de types de média acceptés en réponse à la requête. L'astérisque "*" est utilisée pour grouper des types de média par classe, "*/*" indiquant tous les types de média, et "type/*" indiquant tous les sous-types du type "type". La gamme de types signalée par le client sert essentiellement à limiter les documents à des types acceptables par le client dans le contexte de la requête formulée.

D.2.2 Accept-Charset

Le champ d'en-tête Accept-Charset (requête) peut être utilisé pour former une liste de tables de caractères préférentielles, autres que les tables US-ASCII et ISO-8859-1 utilisées par défaut. Ce champ permet à un client de signaler à un serveur qu'il sait accepter d'autres représentations de texte que les représentations par défaut, et est donc en mesure d'exploiter des documents encodés avec ces tables, si le serveur sait les transmettre.

D.2.3 Accept-Encoding

Le champ d'en-tête Accept-Encoding (requête) est similaire au champ Accept, mais restreint la gamme de valeurs Content-Coding attendues dans une réponse.

D.2.4 Accept-Language

Le champ d'en-tête Accept-Encoding (requête) est similaire au champ Accept, mais restreint la gamme de langues attendues dans une réponse.

D.2.5 Content-Language

Le champ d'en-tête Content-Language (entité) décrit la ou les langues naturelles des destinataires potentiels du corps d'entité. Notez que ceci n'a aucun lien avec les langues utilisées dans l'ensemble de l'entité.

D.2.6 Link

Le champ d'en-tête Link (entité) permet de décrire les liaisons entre l'entité jointe et d'autres ressources. Une entité peut se voir attribuer plusieurs valeurs pour le champ Link. Un champ Link, au niveau métainformation, indique typiquement des relations du type dépendance hiérarchique ou des chemins d'accès.

D.2.7 MIME-Version

Les messages HTTP peuvent inclure un champ unique d'en-tête générale indiquant quelle version du protocole MIME a été employé pour construire le message. L'utilisation de ce champ MIME-Version, tel que décrit par la RFC 1521 [5], peut indiquer si le message est compatible MIME. Malheureusement, certaines premières implémentations de serveurs HTTP/1.0 émettent ce champ sans réserve. Il est conseillé de l'ignorer.

D.2.8 Retry-After

Le champ d'en-tête Retry-After (réponse) peut être utilisé dans une réponse 503 (service indisponible) dans le but d'indiquer pendant combien de temps (estimé) le service restera indisponible aux requêtes du client. La valeur de ce champ peut être une date HTTP ou un nombre entier de secondes (en décimal) à partir de l'heure de la réponse.

D.2.9 Title

Le champ d'en-tête Title est purement informatif et donne le titre de l'entité.

D.2.10 URI

Le champ d'en-tête URI (entité) peut contenir toute ou partie des Uniform Resource Identifiers (Section 3.2) par laquelle la ressource définie par l'URI-visée peut être identifiée. Aucune garantie n'est cependant donnée que la ressource soit effectivement accessible par l'une des URI spécifiées.