Groupe de travail Réseau
Request for Comments : 2616
RFC rendue obsolète : 2068
Catégorie : En cours de normalisation juin 1999

Traduction Claude Brière de L'Isle mars 2007

R. Fielding, UC Irvine
J. Gettys, Compaq/W3C
J. Mogul, Compaq
H. Frystyk, W3C/MIT
L. Masinter, Xerox
P. Leach, Microsoft
T. Berners-Lee, W3C/MIT

Protocole de transfert Hypertexte -- HTTP/1.1

Statut de ce mémo
Le présent document spécifie un protocole de normalisation Internet pour la communauté de l'Internet, qui appelle à la discussion et à des suggestions pour son amélioration. Prière de se reporter à l'édition en cours des "Normes de protocole officielles de l'Internet" (STD 1) sur l'état de la normalisation et le statut de ce protocole. La distribution du présent mémo n'est soumise à aucune restriction.

Résumé
Le protocole de transfert hypertexte (HTTP, *Hypertext Transfer Protocol*) est un protocole de niveau application pour les systèmes d'information hypermédia distribués collaboratifs. C'est un protocole générique, sans états, qui peut être utilisé pour de nombreuses tâches autres que l'hypertexte, telles que les systèmes de gestion de serveurs de noms et d'objets distribués, à travers l'extension de ses méthodes de demande, ses codes d'erreur et ses en-têtes [47]. Une caractéristique de HTTP est l'identification et la négociation de représentations de données, permettant aux systèmes d'être construits indépendamment des données à transférer.

HTTP a été utilisé par l'initiative d'informations mondiales du World-Wide Web depuis 1990. La présente spécification définit le protocole désigné sous le nom de "HTTP/1.1", qui est une mise à jour de la RFC 2068 [33].

Table des matières

1 Introduction

1.1 Objet

Le protocole de transfert hypertexte (HTTP, *Hypertext Transfer Protocol)* est un protocole de niveau application pour les systèmes d'information multimédia distribués et collaboratifs. HTTP a été utilisé par l'initiative mondiale d'informations de la Toile (*World-Wide Web*) depuis 1990. La première version de HTTP, portant la référence HTTP/0.9, était un simple protocole pour le transfert de données brutes sur l'Internet. HTTP/1.0, tel que défini par la RFC 1945 [6], améliorait le protocole en permettant que les messages soient au format de messages du style MIME, contenant des méta informations sur les données transférées et des modificateurs de la sémantique des demandes/réponses. Cependant, HTTP/1.0 ne prenait pas suffisamment en considération les effets des mandataires hiérarchiques, de la mise en antémémoire, du besoin de connexions permanentes, ou des hôtes virtuels. De plus, la prolifération d'applications incomplètement mises en œuvre s'intitulant elles-mêmes "HTTP/1.0" a nécessité le changement de la version du protocole afin que deux applications communicantes déterminent chacune les vraies capacités de l'autre.

La présente spécification définit le protocole connu sous le nom de "HTTP/1.1". Ce protocole inclut des exigences plus strictes que HTTP/1.0 afin d'assurer une mise en œuvre fiable de ses caractéristiques.

Les systèmes d'informations pratiques exigent plus de fonctionnalités que la simple restitution, y compris la recherche, la mise à jour de l'extrémité frontale, et l'annotation. HTTP permet un ensemble ouvert de méthodes et d'en-têtes qui indiquent l'objet d'une demande [47]. Il se construit sur une discipline de références fournies par l'identifiant universel de ressource (URI) [3], comme une localisation (URL) [4] ou un nom (URN) [20], pour indiquer la ressource à laquelle une méthode est à appliquer. Les messages sont passés dans un format similaire à celui utilisé par la messagerie Internet [9] comme défini par les extensions multi-objets de messagerie Internet (MIME, *Multipurpose Internet Mail Extensions*) [7].

HTTP est aussi utilisé comme protocole générique pour les communications entre agents d'utilisateur et mandataires/passerelles vers d'autres systèmes Internet, y compris ceux pris en charge par les protocoles SMTP [16], NNTP [13], FTP [18], Gopher [2], et WAIS [10]. De cette façon, HTTP permet un accès hypermédia de base aux ressources disponibles à partir d'applications diverses.

1.2 Exigences

Les mots clé "DOIT", "NE DOIT PAS", "EXIGÉ", "DEVRAIT", "NE DEVRAIT PAS", "RECOMMANDÉ", "PEUT", et "FACULTATIF" dans le présent document sont à interpréter comme décrit dans le BCP 14 [RFC2119] [34].

Une mise en œuvre n'est pas conforme si elle échoue à satisfaire une ou plusieurs exigences de niveau DOIT ou EXIGÉ pour les protocoles qu'elle met en œuvre. Une mise en œuvre qui satisfait toutes les exigences de niveau DOIT ou EXIGÉ et toutes les exigences de niveau DEVRAIT pour ses protocoles est dite "inconditionnellement conforme" ; celle qui satisfait toutes les exigences de niveau DOIT mais pas toutes les exigences de niveau DEVRAIT pour ses protocoles est dite "conditionnellement conforme".

1.3 Terminologie

La présente spécification utilise un certain nombre de termes pour se référer aux rôles joués par les participants à la communication HTTP et à ses objets.

`connexion`
Un circuit virtuel de couche transport établi entre deux programmes pour les besoins de la communication.

`message`
Unité de base de la communication HTTP, consistant en une séquence structurée d'octets satisfaisant à la syntaxe définie à la section 4 et transmise via la connexion.

`demande`
Un message de demande HTTP, comme défini à la section 5.

réponse
Un message de réponse HTTP, comme défini à la section 6.

ressource
Un objet ou service de données de réseau qui peut être identifié par un URI, comme défini au paragraphe 3.2. Les ressources peuvent être disponibles dans plusieurs représentations (par exemple, plusieurs langages, formats de données, taille, et résolutions) ou varier dans d'autres façons.

entité
Ce sont les informations transférées comme charge utile d'une demande ou réponse. Une entité consiste en méta informations sous la forme de champs d'en-tête d'entité et en contenu sous la forme d'un corps d'entité, comme décrit à la section 7.

représentation
C'est une entité incluse avec une réponse qui est sujette à négociation de contenu, comme décrit à la section 12. Il peut exister plusieurs représentations associées à un état de réponse particulier.

négociation de contenu
Mécanisme pour choisir la représentation appropriée lors de la satisfaction d'une demande, comme décrit à la section 12. La représentation des entités dans toute réponse peut être négociée (y compris les réponses d'erreur).

variante
Une ressource peut avoir une, ou plus d'une, représentations associées à tout instant donné. Chacune de ces représentations est appelée une 'variante'. L'utilisation du terme 'variante' n'implique pas nécessairement que la ressource soit soumise à une négociation de contenu.

client
Un programme qui établit des connexions pour les besoins de l'envoi des demandes.

agent d'utilisateur
Le client qui initie une demande. Ce sont souvent des navigateurs, des éditeurs, des robots araignées (robots de traversée de la Toile), ou autres outils d'utilisateur terminal.

serveur
Un programme d'application qui accepte les connexions afin de servir des demandes en renvoyant des réponses. Tout programme donné peut être capable d'être à la fois un client et un serveur ; notre utilisation de ces termes se réfère seulement au rôle effectué par le programme pour une connexion particulière, plutôt qu'aux capacités du programme en général. De même, tout serveur peut agir comme un serveur d'origine, un mandataire, une passerelle, ou un tunnel, changeant de comportement sur la base de la nature de chaque demande.

serveur d'origine
Le serveur sur lequel une ressource donnée réside ou est à créer.

mandataire *(proxy)*
Un programme intermédiaire qui agit à la fois comme un serveur et un client dans le but de faire des demandes au nom d'autres clients. Les demandes sont servies en interne ou en les transmettant, avec une éventuelle traduction, à d'autres serveurs. Un mandataire DOIT mettre en œuvre à la fois les exigences de client et de serveur de la présente spécification. Un "mandataire transparent" est un mandataire qui ne modifie pas la demande ou réponse au delà de ce qui est requis pour l'authentification et l'identification par le mandataire. Un "mandataire non transparent" est un mandataire qui modifie la demande ou réponse afin de fournir des services supplémentaires à l'agent d'utilisateur, tels que des services d'annotation de groupe, de transformation de type de support, de réduction de protocole, ou de filtrage de l'anonymat. Excepté lorsque le comportement transparent ou non transparent est explicitement déclaré, les exigences de mandataire HTTP s'appliquent aux deux types de mandataires.

passerelle
Serveur qui agit comme intermédiaire pour un autre serveur. À la différence d'un mandataire, une passerelle reçoit des demandes comme si elle était le serveur d'origine pour la ressource demandée ; le client demandeur peut n'être pas avisé qu'il communique avec une passerelle.

tunnel
Un programme intermédiaire qui agit comme un relais aveugle entre deux connexions. Une fois actif, un tunnel n'est pas considéré comme partie à la communication HTTP, bien que le tunnel puisse avoir été initialisé par une demande HTTP. Le tunnel cesse d'exister lorsque les deux extrémités de la connexion relayée sont closes.

antémémoire
Mémoire locale d'un programme de messages de réponse et le sous-système qui contrôle sa mémorisation, restitution et suppression de messages. Une antémémoire stocke les réponses éligibles afin de réduire le temps de réponse et la consommation de bande passante du réseau sur les demandes futures, équivalentes. Tout client ou serveur peut inclure une antémémoire, bien qu'elle ne puisse pas être utilisée par un serveur qui agit comme un tunnel.

mémorisable
Une réponse est mémorisable si une antémémoire est autorisée à stocker une copie du message de réponse pour l'utiliser à répondre à des demandes ultérieures. Les règles de détermination de la possibilité de mémoriser les réponses HTTP sont définies à la section 13. Même si une ressource est mémorisable, il peut y avoir des contraintes supplémentaires sur la question de savoir si une antémémoire peut utiliser la copie mémorisée pour une demande particulière.

première main
Une réponse est de première main si elle vient directement et sans délai inutile du serveur d'origine, éventuellement via un ou plusieurs mandataires. Une réponse est aussi de première main si sa validité vient d'être vérifiée directement avec le serveur d'origine.

heure d'expiration explicite
C'est l'heure à laquelle le serveur d'origine a prévu qu'une entité ne devrait plus être retournée par une antémémoire sans autre validation.

heure d'expiration euristique
Heure d'expiration allouée par une antémémoire lorsque aucune heure d'expiration explicite n'est disponible.

âge
L'âge d'une réponse est le temps écoulé depuis qu'elle a été envoyée, ou validée avec succès avec le serveur d'origine.

fraîcheur de durée de vie
Temps entre la génération d'une réponse et son heure d'expiration.

fraîcheur
Une réponse est fraîche si son âge n'a pas encore excédé sa fraîcheur de durée de vie.

périmée
Une réponse est périmée si son âge a dépassé sa fraîcheur de durée de vie.

sémantiquement transparent
Une antémémoire se comporte d'une façon "sémantiquement transparente", par rapport à une réponse particulière, lorsque son utilisation n'affecte ni le client demandeur ni le serveur d'origine, excepté pour améliorer la performance. Lorsque une antémémoire est sémantiquement transparente, le client reçoit exactement la même réponse (excepté pour les en-têtes de bond par bond) qu'il aurait reçue si sa demande avait été traitée directement par le serveur d'origine.

valideur
Élément de protocole (par exemple, une étiquette d'entité ou une heure de dernière modification) qui est utilisé pour trouver si une entrée d'antémémoire est une copie équivalente d'une entité.

vers l'amont/vers l'aval
Vers l'amont et vers l'aval décrivent les flux de messages : tous les messages vont de l'amont vers l'aval.

entrant/sortant
Entrant et sortant se réfère aux chemins de la demande et de la réponse pour les messages : "entrant" signifie "voyager vers le serveur d'origine", et "sortant" signifie "voyager vers l'agent d'utilisateur"

1.4 Fonctionnement global

Le protocole HTTP est un protocole de demande/réponse. Un client envoie une demande au serveur sous la forme d'une méthode de demande, d'un URI, et d'une version de protocole, suivis par un message de style MIME contenant des modificateurs de demande, des informations de client, et éventuellement un contenu de corps sur une connexion avec un serveur. Le serveur répond par une ligne d'état, incluant la version de protocole du message et un code de succès ou d'erreur, suivi par un message de style MIME contenant des informations de serveur, des méta-informations d'entité, et éventuellement un contenu de corps d'entité. La relation entre HTTP et MIME est décrite à l'appendice 19.4.

La plupart des communications HTTP sont initialisées par un agent d'utilisateur et consistent en une demande d'être appliquées à une ressource sur un serveur d'origine. Dans le cas le plus simple, ceci peut être accompli via une seule connexion (v) entre l'agent d'utilisateur (UA) et le serveur d'origine (O).

```
   chaîne de demande ------------------->
UA -------------------v------------------- O
   <----------------------- chaîne de réponse
```

Une situation plus compliquée survient lorsque un ou plusieurs intermédiaires sont présents dans la chaîne de demande/réponse. Il y a trois formes communes d'intermédiaire : mandataire, passerelle, et tunnel. Un mandataire est un agent de transmission, qui reçoit des demandes pour un URI dans sa forme absolue, réécrit tout ou partie du message, et transmet la demande reformatée vers le serveur identifié par l'URI. Une passerelle est un agent receveur, qui agit comme une couche au-dessus d'un ou d'autres serveurs et, si nécessaire, traduit les demandes au protocole du serveur sous-jacent. Un tunnel agit comme un point de relais entre deux connexions sans changer les messages ; les tunnels sont utilisés lorsque la communication doit passer à travers un intermédiaire (comme un pare-feu) même lorsque l'intermédiaire ne peut pas comprendre le contenu des messages.

```
chaîne de demande ---------------------------------------->
UA -----v----- A -----v----- B -----v----- C -----v----- O
   <------------------------------------ chaîne de réponse
```

La figure ci-dessus montre trois intermédiaires (A, B, et C) entre l'agent d'utilisateur et le serveur d'origine. Une demande ou message de réponse qui voyage à travers toute la chaîne va passer à travers quatre connexions distinctes. Cette distinction est importante parce que certaines options de communication HTTP peuvent ne s'appliquer qu'à la connexion avec le plus proche voisin, non-tunnel, seulement aux points d'extrémité de la chaîne, ou à toutes les connexions le long de la chaîne. Bien que le diagramme soit linéaire, chaque participant peut être engagé dans plusieurs communications simultanées. Par exemple, B peut être en train de recevoir des demandes de nombreux clients autres que A, et/ou transmettre des demandes à des serveurs autres que C, au moment même où il traite la demande de A.

Toute partie à la communication qui n'agit pas comme un tunnel peut employer une antémémoire interne pour le traitement des demandes. L'effet d'une antémémoire est que la chaîne des demandes/réponses est raccourcie si un des participants le long de la chaîne a une réponse en antémémoire applicable à cette demande. Ce qui suit illustre la chaîne résultante si B a une copie en antémémoire d'une réponse précédente de la part de O (via C) pour une demande qui n'avait pas été mise en antémémoire par UA ou A.

```
chaîne de demande ---------------------------------->
UA -----v----- A -----v----- B - - - - - - C - - - - - - - O
 <------------------------------------ chaîne de réponse
```

Toutes les réponses ne peuvent pas être mises de façon utile en antémémoire, et certaines demandes peuvent contenir des modificateurs qui imposent des exigences particulières sur le comportement de l'antémémoire. Les exigences de HTTP pour le comportement de l'antémémoire et des réponses éligibles sont définies à la section 13.

En fait, une grande variété d'architectures et de configurations d'antémémoires et de mandataires est en cours d'expérimentation ou de développement à travers la Toile mondiale. Ces systèmes incluent des hiérarchies nationales d'antémémoires mandataires pour économiser la bande passante transocéanique, des systèmes qui font une diffusion générale ou une diffusion groupée des entrées d'antémémoire, des organisations qui

distribuent des sous-ensembles de données en antémémoire via CD-ROM, et ainsi de suite. Les systèmes HTTP sont utilisés dans des intranets d'entreprise sur des liaisons à forte bande passante, et pour des accès via des PDA ayant des liaisons radio à faible puissance et une connectivité intermittente. Le but de HTTP/1.1 est de prendre en charge la grande diversité de configurations déjà déployée tout en introduisant les éléments de protocole qui satisfont les besoins de ceux qui construisent des applications de la Toile qui exigent une haute fiabilité et, en cas d'échec, des indications fiables de cet échec.

Les communications HTTP ont généralement lieu sur des connexions TCP/IP. L'accès par défaut est TCP 80 [19], mais d'autres ports peuvent être utilisés. Cela n'empêche pas HTTP d'être mis en œuvre par dessus tout autre protocole sur l'Internet, ou sur d'autres réseaux. HTTP présuppose seulement un transport fiable ; tout protocole qui fournit de telles garanties peut être utilisé ; la transposition des structures de demande et réponses HTTP/1.1 en unités de données de transport du protocole en question sort du domaine d'application de la présente spécification.

Dans HTTP/1.0, la plupart des mises en œuvre utilisent une nouvelle connexion pour chaque échange de demande/réponse. Dans HTTP/1.1, une connexion peut être utilisée pour un ou plusieurs échanges de demandes et réponses, bien que les connexions puissent être closes pour des raisons diverses (voir au paragraphe 8.1).

2 Conventions de notation et grammaire générique

2.1 BNF augmenté

Tous les mécanismes spécifiés dans le présent document sont décrits à la fois en prose et en forme Backus-Naur (BNF) augmenté semblable à celle qui est utilisée par la RFC 822 [9]. Les développeurs devront être familiers avec la notation afin de comprendre la présente spécification. Le BNF augmenté comporte les constructions suivantes :

`name = definition`
Le nom d'une règle est simplement le nom lui-même (sans aucune inclusion dans des crochets angulaires "<" et ">") et il est séparé de sa définition par le caractère égal "=". L'espace n'est significative que dans le cas où des continuations de lignes sont utilisées pour indiquer qu'une définition de règle s'étend sur plus d'une ligne. Certaines règles de base sont en majuscules, comme SP, LWS, HT, CRLF, DIGIT, ALPHA, etc. Les crochets angulaires sont utilisés au sein des définitions chaque fois que leur présence va faciliter l'identification de l'utilisation des noms de règles.

`"literal"`
Des guillemets entourent le texte littéral. Sauf mention contraire, le texte est insensible à la casse.

`rule1 | rule2`
Les éléments séparé par une barre ("|") sont alternatifs, par exemple, "oui | non" acceptera oui ou non.

`(rule1 rule2)`
Les éléments entre parenthèses sont traités comme un seul élément. Et donc, "(elem (foo | bar) elem)" permet les séquences de jetons "elem foo elem" et "elem bar elem".

`*rule`
Le caractère "*" précédant un élément indique la répétition. La forme complète est "<n>*<m>element" indiquant au moins <n> et au plus <m> occurrences de l'élément. Les valeurs par défaut sont 0 et l'infini, de sorte que "*(element)" permet tout nombre, y compris zéro ; "1*element" exige au moins un ; et "1*2element" permet un ou deux.

`[rule]`
Des crochets angulaires entourent les éléments facultatifs ; "[foo bar]" est équivalent à "*1(foo bar)".

`n rule`
Spécifique de la répétition : "<n>(element)" est équivalent à "<n>*<n>(element)"; c'est-à-dire, exactement <n> occurrences de (element). Et donc 2DIGIT est un nombre à deux chiffres, et 3ALPHA est une chaîne de trois caractères alphabétiques.

`#rule`

Une construction "#" est définie, comme "*", pour définir des listes d'éléments. La forme complète est "<n>#<m>element" indiquant au moins <n> et au plus <m> éléments, chacun séparé par une ou plusieurs virgules (",") et des espaces blanches linéaires (LWS, *linear white space*) OPTIONAL *(facultatif)*. Cela rend la forme usuelle des listes très facile ; une règle comme (*LWS element *(*LWS "," *LWS element)) peut être indiquée comme 1#element
Chaque fois que cette construction est utilisée, des éléments nuls sont admis, mais ne contribuent pas au compte des éléments présents. C'est-à-dire que, "(element), , (element) " est permis, mais ne compte que comme deux éléments. Donc, lorsque au moins un élément est exigé, au moins un élément non nul DOIT être présent. Les valeurs par défaut sont 0 et l'infini de sorte que "#element" permet tout nombre, y compris zéro; "1#element" exige au moins un, et "1#2element" permet un ou deux.

`; comment`
Un point-virgule, établi à quelque distance à droite du texte de la règle, débute un commentaire qui continue jusqu'à la fin de ligne. C'est une façon simple d'inclure des notes utiles en parallèle avec les spécifications.

`*LWS impliqué`
La grammaire décrite par la présente spécification est fondée sur le mot. Sauf mention contraire, les espaces blanches linéaires (LWS) peuvent être incluses entre deux mots adjacents quelconques (jeton ou chaîne entre guillemets) et entre des mots et séparateurs adjacents, sans changer l'interprétation d'un champ. Au moins un délimiteur (LWS et/ou séparateurs) DOIT exister entre deux jetons quelconques (voir la définition de "jeton" ci-dessous), car autrement, ils seraient interprétés comme un seul jeton.

2.2 Règles de base

Les règles qui suivent sont utilisées tout au long de la présente spécification pour décrire les constructions de base de l'analyse grammaticale. Le jeu de caractères codés en US-ASCII est défini par ANSI X3.4-1986 [21].

OCTET	<toute séquence de 8 bits de données>
CHAR	<tout caractère US-ASCII (octets 0 - 127)>
UPALPHA	<toute lettre majuscule US-ASCII "A".."Z">
LOALPHA	<toute lettre minuscule US-ASCII "a".."z">
ALPHA	UPALPHA | LOALPHA
DIGIT	<tout chiffre US-ASCII "0".."9">
CTL	<tout caractère de contrôle US-ASCII (octets 0 - 31) et DEL (127)>
CR	<US-ASCII CR, retour chariot (13)>
LF	<US-ASCII LF, saut à la ligne (10)>
SP	<US-ASCII SP, espace (32)>
HT	<US-ASCII HT, tabulation horizontale (9)>
<">	<guillemets US-ASCII (34)>

HTTP/1.1 définit la séquence CR LF comme le marquage de fin de ligne pour tous les éléments de protocole excepté le corps d'entité (voir au paragraphe 19.3 de l'Appendice les tolérances d'application). Le marqueur de fin de ligne au sein d'un corps d'entité est défini par son type de support associé, comme décrit au paragraphe 3.7.

```
CRLF  = CR LF
```

Les valeurs de champ d'en-tête HTTP/1.1 peuvent être étalées sur plusieurs lignes si la ligne de continuation commence avec une espace ou une tabulation horizontale. Toutes les espaces blanches linéaires, y compris les sauts de ligne, ont la même sémantique que SP. Un receveur PEUT remplacer toute espace blanche linéaire par un seul SP avant d'interpréter la valeur de champ ou de transmettre le message vers l'aval.

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

La règle TEXT n'est utilisée que pour les contenus de champ descriptifs et les valeurs qui ne sont pas destinées à être interprétées par l'analyseur de message. Les mots de *TEXT PEUVENT contenir des caractères provenant de jeux de caractères autres que ISO-8859-1 [22] s'ils sont codés conformément aux règles de la RFC 2047 [14].

```
TEXT  = <tout OCTET excepté les CTL, mais y compris LWS>
```

Un CRLF n'est admis dans la définition de TEXT qu'au titre de la continuation d'un champ d'en-tête. On s'attend à ce que l'espace LWS soit remplacée par une seule SP avant l'interprétation de la valeur de TEXT.

Les caractères numériques hexadécimaux sont utilisés dans plusieurs éléments de protocole.

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

De nombreuses valeurs de champ d'en-tête HTTP/1.1 consistent en mots séparés par des LWS ou des caractères spéciaux. Ces caractères spéciaux DOIVENT être dans une chaîne entre guillemets pour être utilisés au sein d'une valeur de paramètre (comme défini au paragraphe 3.6).

```
jeton  = 1*<tout CHAR excepté CTLs ou séparateurs>
séparateurs  = "(" | ")" | "<" | ">" | "@" | "," | ";" | ":" | "\" | <"> | "/" | "[" |
               "]" | "?" | "=" | "{" | "}" | SP | HT
```

Des commentaires peuvent être inclus dans certains champs d'en-tête HTTP en entourant le texte du commentaire de parenthèses. Les commentaires ne sont admis que dans les champs qui contiennent "comment" au titre de leur définition de valeur de champ. Dans tous les autres champs, les parenthèses sont considérées comme faisant partie de la valeur du champ.

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

Une chaîne de texte est analysée comme un seul mot si elle est encadrée de doubles guillemets.

```
quoted-string  = ( <"> *(qdtext | quoted-pair ) <"> )
qdtext  = <tout TEXT excepté <">>
```

Le caractère barre oblique inversée ("\") PEUT être utilisé comme un mécanisme de citation à un seul caractère mais seulement dans une chaîne entre guillemets et dans un commentaire.

```
quoted-pair  = "\" CHAR
```

3 Paramètres de protocole

3.1 Version HTTP

HTTP utilise un schéma de numérotation "<major>.<minor>" pour indiquer les versions du protocole. La politique de versions du protocole est destinée à permettre à l'envoyeur d'indiquer le format d'un message et sa capacité à comprendre la suite de la communication HTTP, plutôt que les caractéristiques obtenues via cette communication. Aucun changement n'est fait au numéro de version pour l'ajout de composants de message qui n'affectent pas le comportement de la communication ou qui ajoutent seulement des valeurs de champs extensibles. Le numéro <minor> est incrémenté lorsque les changements apportés au protocole ajoutent des caractéristiques qui ne changent pas l'algorithme général d'analyse du message, mais qui peuvent ajouter à la sémantique du message et impliquer des capacités additionnelles de l'envoyeur. Le numéro <majeur> est incrémenté lorsque le format d'un message au sein du protocole est changé. Voir des explications plus complètes dans la RFC 2145 [36].

La version d'un message HTTP est indiquée par un champ HTTP-Version dans la première ligne du message.

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

Noter que les numéros majeurs et mineurs DOIVENT être traités comme des entiers séparés et que chacun PEUT être incrémenté de plus d'un chiffre. Et donc, HTTP/2.4 est une version antérieure à HTTP/2.13, qui à son tour est antérieure à HTTP/12.3. Les zéros en tête DOIVENT être ignorés par les receveurs et NE DOIVENT PAS être envoyés.

Une application qui envoie une demande ou message de réponse qui inclut une HTTP-Version de "HTTP/1.1" DOIT être au moins conditionnellement conforme à cette spécification. Les applications qui sont au moins conditionnellement conformes à cette spécification DEVRAIENT utiliser une HTTP-Version de "HTTP/1.1" dans leurs messages, et DOIVENT le faire pour tout message qui n'est pas compatible avec HTTP/1.0. Pour des précisions sur le moment où envoyer des valeurs de HTTP-Version spécifiques, voir la RFC 2145 [36].

La version HTTP d'une application est la version HTTP de plus haut chiffre pour laquelle l'application est au moins conditionnellement conforme.

Les applications de mandataires et de passerelles doivent être vigilantes lors de la transmission de messages de versions de protocole différentes de celle de l'application. Comme la version du protocole indique les capacités de protocole de l'envoyeur, un mandataire/passerelle NE DOIT PAS envoyer un message avec un indicateur de version qui soit supérieur à sa version réelle. Si une demande d'une version plus récente est reçue, le mandataire/passerelle DOIT soit dégrader la version de demande, soit répondre par une erreur, soit passer au comportement de tunnel.

Du fait des problèmes d'interopérabilité avec les mandataires HTTP/1.0 découverts depuis la publication de la RFC 2068 [33], les mandataires à antémémoire DOIVENT, les passerelles PEUVENT, et les tunnels NE DOIVENT PAS, améliorer la demande à la plus haute version qu'ils acceptent. La réponse du mandataire/passerelle à cette demande DOIT être de la même version majeure que la demande.

Note : La conversion entre versions de HTTP peut impliquer des modifications de champs d'en-tête exigées ou interdites par les versions impliquées.

3.2 Identifiants universels de ressource

Les URI ont été appelés de nombreuses façons : adresses WWW, identifiants universels de documents, identifiants de ressource universels [3], et finalement sont la combinaison des localisateurs uniformes de ressource (URL) [4] et des noms universels de ressource (URN) [20]. Pour ce qui concerne HTTP, les identifiants universels de ressource sont simplement des chaînes formatées qui identifient -- via le nom, la localisation, ou toute autre caractéristique -- une ressource.

3.2.1 Syntaxe générale

Les URI dans HTTP peuvent être représentés en forme absolue ou relative par rapport à un URI de base connu [11], selon le contexte de leur utilisation. Les deux formes sont différenciées par le fait que les URI absolus commencent toujours par un nom de schéma suivi par un point. Pour des informations précises sur la syntaxe et la sémantique des URL, voir la RFC 2396 [42] "Identifiants universels de ressource (URI) : Syntaxe générique et sémantique," (qui remplace les RFC 1738 [4] et RFC 1808 [11]). La présente spécification adopte les définitions de "URI-reference", "absoluteURI", "relativeURI", "port", "host","abs_path", "rel_path", et "authority" données dans cette spécification.

Le protocole HTTP ne met aucune limite a priori sur la longueur d'un URI. Les serveurs DOIVENT être capables de traiter les URI quelle que soit la ressource qu'ils desservent, et DEVRAIENT être capables de traiter les URI sans limite de longueur s'ils fournissent des formes fondées sur GET qui pourraient générer de tels URI. Un serveur DEVRAIT retourner un état 414 (Request-URI trop long) si un URI est plus long que ce que le serveur peut traiter (voir au paragraphe 10.4.15).

Note : Les serveurs devraient être vigilants sur la question des longueurs d'URI supérieures à 255 octets, parce que quelques mises en œuvre anciennes de client ou mandataire pourraient ne pas prendre en charge correctement des longueurs.

3.2.2 URL http

Le schéma "http" est utilisé pour localiser des ressources de réseau via le protocole HTTP. Ce paragraphe définit la syntaxe et la sémantique spécifiques du schéma pour les URL http.

```
http_URL = "http:" "//" host [ ":" port ] [ abs_path [ "?" query ]]
```

Si le port est vide ou n'est pas donné, on suppose le port 80. La sémantique est que la ressource identifiée est localisée au serveur qui écoute les connexions TCP sur ce port chez cet hôte, et le Request-URI pour la ressource est abs_path (paragraphe 5.1.2). L'utilisation des adresses IP dans les URL DEVRAIT être évitée chaque fois que possible (voir la RFC 1900 [24]). Si le abs_path n'est pas présent dans l'URL, il DOIT être donné par "/" lorsque utilisé comme un Request-URI pour une ressource (paragraphe 5.1.2). Si un mandataire

reçoit un nom d'hôte qui n'est pas un nom de domaine pleinement qualifié, il PEUT ajouter son domaine au nom d'hôte qu'il a reçu. Si un mandataire reçoit un nom de domaine pleinement qualifié, le mandataire NE DOIT PAS changer le nom d'hôte.

3.2.3 Comparaison d'URI

En comparant deux URI pour décider si ils correspondent ou non, un client DEVRAIT utiliser une comparaison sensible à la casse octet par octet de l'URI tout entier, avec les exceptions suivantes :

- Un port qui est vide ou non donné est équivalent au port par défaut pour cette référence d'URI ;
- Les comparaisons des noms d'hôte DOIVENT être insensibles à la casse ;
- Les comparaisons de noms de schéma DOIVENT être insensibles à la casse ;
- Un abs_path vide est équivalent à un abs_path de "/".

Les caractères autres que ceux dans les ensembles "reserved" et "unsafe" (voir la RFC 2396 [42]) sont équivalents à leur codage ""%" HEX HEX".

Par exemple, les trois URI suivants sont équivalents :

```
http://abc.com:80/~smith/home.html
http://ABC.com/%7Esmith/home.html
http://ABC.com:/%7esmith/home.html
```

3.3 Formats de date/heure

3.3.1 Date complète

Les applications HTTP ont autrefois admis trois formats différents pour la représentation des horodatages :

```
Sun, 06 Nov 1994 08:49:37 GMT  ; RFC 822, mise à jour par la RFC 1123
Sunday, 06-Nov-94 08:49:37 GMT  ; RFC 850, rendue obsolète par la RFC 1036
Sun Nov  6 08:49:37 1994  ; format asctime() de l'ANSI C
```

Le premier format est le préféré comme norme Internet et représente un sous ensemble de longueur fixe de ce qui est défini par la RFC 1123 [8] (une mise à jour de la RFC 822 [9]). Le second format est d'utilisation courante, mais il se fonde sur le format de date obsolète de la RFC 850 [12] et manque des quatre chiffres de l'année. Les clients et serveurs HTTP/1.1 qui analysent la valeur de cette date DOIVENT accepter ces trois formats (pour la compatibilité avec HTTP/1.0) bien qu'ils DOIVENT seulement générer le format de la RFC 1123 pour représenter les valeurs de date HTTP dans les champs d'en-tête. Voir au paragraphe 19.3 pour des précisions.

Note : Les receveurs de valeurs de date sont invités à une certaine robustesse pour accepter des valeurs de date qui peuvent avoir été envoyées par des applications non HTTP, comme c'est parfois le cas lors de la restitution ou l'envoi de messages via des mandataires/passerelles à SMTP ou NNTP.

Tous les horodatages HTTP DOIVENT être représentés en temps moyen de Greenwich (GMT), sans exception. Pour les besoins de HTTP, GMT est exactement égal à l'UTC (Temps universel coordonné). Ceci est indiqué dans les deux premiers formats par l'inclusion de "GMT" comme abréviation en trois lettres pour la zone horaire, et DOIT être supposé lors de la lecture du format asctime. La date HTTP est sensible à la casse et NE DOIT PAS inclure de LWS supplémentaires au delà de ce qui est spécifiquement inclus comme SP dans la grammaire.

HTTP-date	rfc1123-date | rfc850-date | asctime-date
rfc1123-date	wkday "," SP date1 SP time SP "GMT"
rfc850-date	weekday "," SP date2 SP time SP "GMT"
asctime-date	wkday SP date3 SP time SP 4DIGIT
date1 ; jour mois an (par exemple, 02 Jun 1982)	2DIGIT SP month SP 4DIGIT
date2 ; jour-mois-an (par exemple, 02-Jun-82)	2DIGIT "-" month "-" 2DIGIT
date3 ; mois jour (par exemple, Jun 2)	month SP (2DIGIT | (SP 1DIGIT))
time ; 00:00:00 - 23:59:59	2DIGIT ":" 2DIGIT ":" 2DIGIT
wkday	"Mon" | "Tue" | "Wed" | "Thu" | "Fri" | "Sat" | "Sun"
weekday	"Monday" | "Tuesday" | "Wednesday" | "Thursday" | "Friday"

| "Saturday" | "Sunday"
month "Jan" | "Feb" | "Mar" | "Apr" | "May" | "Jun" | "Jul" | "Aug" | "Sep" | "Oct" | "Nov" | "Dec"

Note : lLes exigences de HTTP pour le format d'horodatage ne s'appliquent qu'à leur utilisation dans le flux de protocole. Les clients et serveurs ne sont pas obligés d'utiliser ces formats pour la présentation à l'utilisateur, la journalisation des demandes, etc.

3.3.2 Delta secondes

Certains champs d'en-tête HTTP permettent qu'une valeur d'heure soit spécifiée comme un nombre entier de secondes, représenté en décimal, après l'heure de réception du message.

```
delta-seconds  = 1*DIGIT
```

3.4 Jeux de caractères

HTTP utilise la même définition du terme "jeu de caractères" que ce qui est décrit pour MIME :

Le terme "jeu de caractères" est utilisé dans le présent document pour se référer à une méthode utilisée avec un ou plusieurs tableaux pour convertir une séquence d'octets en une séquence de caractères. Noter qu'une conversion inconditionnelle dans l'autre direction n'est pas exigée, car tous les caractères peuvent n'être pas disponibles dans un jeu de caractères donné et un jeu de caractères peut fournir plus d'une séquence d'octets pour représenter un caractère particulier. Cette définition est destinée à permettre diverses sortes de codages de caractères, depuis les transpositions d'un simple tableau tel que l'US-ASCII jusqu'à des méthodes complexes de commutation de tableaux telles que celles qu'utilisent les techniques de la norme ISO-2022. Cependant, la définition associée au nom de jeu de caractères MIME DOIT pleinement spécifier la transposition à effectuer des octets en caractères. En particulier, l'utilisation d'informations de profilage externes pour déterminer la transposition exacte n'est pas permise.

Note : Cette utilisation du terme "jeu de caractères" est plus couramment utilisée comme "codage de caractères." Cependant, comme HTTP et MIME partagent le même registre, il est important de partager aussi la terminologie.

Les jeux de caractères HTTP sont identifiés par des jetons insensibles à la casse. L'ensemble complet des jetons est défini par le registre IANA Character Set [19].

```
charset = token
```

Bien que HTTP permette qu'un jeton arbitraire soit utilisé comme valeur de charset, tout jeton qui a une valeur prédéfinie au sein du registre IANA Character Set [19] DOIT représenter le jeu de caractères défini par ce registre. Les applications DEVRAIENT limiter leur utilisation des jeux de caractères à celles définies par le registre de l'IANA.

Les développeurs devraient être au courant des exigences de jeu de caractères de l'IETF [38] [41].

3.4.1 Ensemble de caractères absent

Certains logiciels HTTP/1.0 ont interprété incorrectement un en-tête Content-Type sans paramètre charset comme signifiant "au receveur de deviner." Les envoyeurs qui souhaitent combattre ce comportement PEUVENT inclure un paramètre charset même lorsque le charset est ISO-8859-1 et DEVRAIENT le faire lorsque il est clair qu'il n'y aura pas de confusion chez le receveur.

Malheureusement, certains clients HTTP/1.0 plus anciens ne traitaient pas de façon appropriée un paramètre charset explicite. Les receveurs HTTP/1.1 DOIVENT respecter l'étiquette charset fournie par l'envoyeur ; et les agents d'utilisateur qui ont des dispositions pour "deviner" un charset DOIVENT utiliser le charset provenant du champ content-type si ils acceptent ce charset, plutôt que la préférence du receveur, lorsqu'ils affichent un document pour la première fois. Voir au paragraphe 3.7.1.

3.5 Codages de contenu

Les valeurs de codage de contenu indiquent une transformation par codage qui a été ou peut être appliquée à une entité. Les codages de contenu sont principalement utilisés pour permettre de compresser un document ou autrement de le transformer utilement sans perdre l'identité de son type de support sous-jacent et sans perte d'information. Fréquemment, l'entité est mémorisée dans une forme codée, transmise directement, et seulement décodée par le receveur.

```
content-coding  = jeton
```

Toutes les valeurs de content-coding sont insensibles à la casse. HTTP/1.1 utilise des valeurs de content-coding dans les champs d'en-tête Accept-Encoding (paragraphe 14.3) et Content-Encoding (paragraphe 14.11). Bien que la valeur décrive le content-coding, ce qui est le plus important est que cela indique quel mécanisme de décodage sera exigé pour retirer le codage.

L'Autorité d'allocation des numéros de l'Internet (IANA) agit comme registraire pour les jetons de valeur de content-coding. Au départ, le registre contenait les jetons suivants :

gzip
Format de codage produit par le programme de compression de fichiers "gzip" (GNU zip) comme décrit à la RFC 1952 [25]. Ce format est un codage Lempel-Ziv (LZ77) avec un CRC de 32 bits.

compress
Format de codage produit par le programme UNIX commun de compression de fichiers "compress". Ce format est un codage Lempel-Ziv-Welch adaptatif (LZW).
L'utilisation de noms de programmes pour l'identification des formats de codage n'est pas souhaitable et est déconseillée pour les codages futurs. Leur utilisation est représentative de pratiques historiques, mais pas d'une saine conception. Pour la compatibilité avec les mises en œuvre précédentes de HTTP, les applications DEVRAIENT considérer "x-gzip" et "x-compress" comme respectivement équivalentes à "gzip" et "compress".

deflate
Format "zlib" défini dans la RFC 1950 [31] en combinaison avec le mécanisme de compression "deflate" décrit dans la RFC 1951 [29].

identity
Codage par défaut (identité) ; n'utilise aucune transformation. Ce codage de contenu n'est utilisé que dans l'en-tête Accept-Encoding, et NE DEVRAIT PAS être utilisé dans l'en-tête Content-Encoding.

De nouveaux jetons de valeur de content-coding DEVRAIENT être enregistrés ; pour permettre l'interopérabilité entre clients et serveurs, des spécifications d'algorithmes de codage de contenu nécessaires pour la mise en œuvre de nouvelles valeurs DEVRAIENT être disponibles au public et adéquates pour une mise en œuvre indépendante, et se conformer à l'objectif de codage de contenu défini dans ce paragraphe.

3.6 Codages de transferts

Les valeurs de transfer-coding *(codage de transfert)* sont utilisées pour indiquer qu'une transformation de codage a été, peut être, ou doit être appliquée à un corps d'entité afin de s'assurer d'un "transport sûr" à travers le réseau. Ceci diffère d'un codage de contenu en ce que le codage de transfert est une propriété du message, et non de l'entité originelle.

```
transfer-coding  = "chunked" | transfer-extension
transfer-extension  = jeton *( ";" paramètre )
```

Les paramètres sont sous forme de paires d'attribut/valeur.

```
paramètre  = attribut "=" valeur
attribut   = jeton
valeur     = jeton | quoted-string
```

Toutes les valeurs de codage de transfert sont insensibles à la casse. HTTP/1.1 utilise les valeurs de codage de transfert dans le champ d'en-tête TE (paragraphe 14.39) et dans le champ d'en-tête Transfer-Encoding (paragraphe 14.41).

Chaque fois qu'un codage de transfert est appliqué à un corps de message, l'ensemble des codages de transfert DOIT inclure "chunked" *(fragmenté)*, sauf si le message est terminé par la clôture de la connexion. Lorsque le transfert de codage "chunked" est utilisé, il DOIT être le dernier codage de transfert appliqué au corps de message. Le codage de transfert "chunked" NE DOIT PAS être appliqué plus d'une fois à un corps de message. Ces règles permettent au receveur de déterminer la longueur du transfert du message (paragraphe 4.4).

Les codages de transfert sont analogues aux valeurs de Content-Transfer-Encoding de MIME [7], qui ont été conçues pour permettre un transport de données binaires sûr par un service de transport à 7 bits. Cependant, le transport sûr a un centre d'intérêt différent pour un protocole de transfert à 8 bits. Dans HTTP, la seule caractéristique incertaine des corps de message est la difficulté de déterminer la longueur exacte du corps (paragraphe 7.2.2), ou le souhait de chiffrer des données sur un transport partagé.

L'Autorité d'allocation des numéros Internet (IANA) agit comme registraire des jetons de valeurs de codage de transfert. À l'origine, le registre contenait les jetons suivants : "chunked" (paragraphe 3.6.1), "gzip" (paragraphe 3.5), "compress" (paragraphe 3.5), et "deflate" (paragraphe 3.5).

De nouveaux jetons de valeur de codage de transfert DEVRAIENT être enregistrés de la même façon que pour les nouveaux jetons de valeur codage de contenu (paragraphe 3.5).

Un serveur qui reçoit un corps d'entité avec un codage de transfert qu'il ne comprend pas DEVRAIT retourner 501 (Non mis en œuvre), et clore la connexion. Un serveur NE DOIT PAS envoyer de codages de transfert à un client HTTP/1.0.

3.6.1 Codage de transfert fractionné

Le codage fractionné modifie le corps d'un message afin de le transférer comme une série de fragments, ayant chacun son propre indicateur de taille, suivi par un en-queue *(trailer)* FACULTATIF *(OPTIONAL)* contenant des champs d'en-tête d'entité. Cela permet de transférer des contenus produits de façon dynamique avec les informations nécessaires pour que le receveur vérifie qu'il a bien reçu tout le message.

```
Chunked-Body   = *chunk
    last-chunk
  trailer
CRLF

chunk  = chunk-size [ chunk-extension ] CRLF
chunk-data CRLF
chunk-size  = 1*HEX
last-chunk  = 1*("0") [ chunk-extension ] CRLF

chunk-extension= *( ";" chunk-ext-name [ "=" chunk-ext-val ] )
chunk-ext-name  = token
chunk-ext-val  = token | quoted-string
chunk-data  = chunk-size(OCTET)
trailer  = *(entity-header CRLF)
```

Le champ chunk-size *(taille de fragment)* est une chaîne de chiffres hexadécimaux qui indique la taille du fragment. Le codage fragmenté se termine par n'importe quel fragment dont la taille est zéro, suivi par l'en-queue, qui se termine par une ligne vide.

L'en-queue permet à l'envoyeur d'inclure des champs d'en-tête HTTP supplémentaires à la fin du message. Le champ d'en-tête Trailer peut être utilisé pour indiquer quels champs d'en-tête sont inclus dans un en-queue (voir au paragraphe 14.40).

Un serveur qui utilise un codage de transfert fractionné dans une réponse NE DOIT PAS utiliser l'en-queue pour un champ d'en-tête sauf si au moins une des conditions suivantes est vraie :

a) la demande comporte un champ d'en-tête TE qui indique que "trailers" est acceptable dans le codage de transfert de la réponse, comme décrit au paragraphe 14.39, ou

b) le serveur est le serveur d'origine pour la réponse, le champ trailer consiste uniquement en méta données facultatives, et le receveur pourrait utiliser le message (d'une façon acceptable pour le serveur d'origine)

sans recevoir ces métadonnées. En d'autres termes, le serveur d'origine est d'accord pour accepter la possibilité que le champ trailer soit éliminé sans formalité le long du chemin vers le client.

Cette exigence empêche l'échec d'interopérabilité lorsque le message doit être reçu par un mandataire HTTP/1.1 (ou plus) et retransmis à un receveur HTTP/1.0. Elle évite une situation où la conformité avec le protocole aurait nécessité une éventuelle mémoire tampon infinie chez le mandataire.

Un exemple de traitement pour le décodage d'un Chunked-Body (corps fractionné) est présenté à l'Appendice 19.4.6.

Toutes les applications HTTP/1.1 DOIVENT être capables de recevoir et décoder le codage de transfert "fractionné", et DOIVENT ignorer les extensions chunk-extension qu'elles ne comprennent pas.

3.7 Types de support

HTTP utilise les types de support Internet [17] dans les champs d'en-tête Content-Type (paragraphe 14.17) et Accept (paragraphe 14.1) afin de fournir une typologie de données et une négociation de type ouvertes et extensibles.

```
media-type  = type "/" subtype *( ";" parameter )
type        = token
subtype     = token
```

Des paramètres PEUVENT suivre le type/sous-type sous la forme de paires d'attribut/valeur (comme défini au paragraphe 3.6).

Les noms d'attribut de type, sous-type, et de paramètre sont insensibles à la casse. Les valeurs de paramètres peuvent être ou non sensibles à la casse, selon la sémantique du nom du paramètre. Les espaces linéaires (LWS) NE DOIVENT PAS être utilisées entre le type et le sous-type, ni entre un attribut et sa valeur. La présence ou l'absence d'un paramètre peut être significative pour le traitement d'un type de support, selon sa définition au sein du registre de type de support.

Noter que certaines anciennes applications HTTP ne reconnaissent pas les paramètres de type de support. Lors de l'envoi de données à d'anciennes applications HTTP, les mises en œuvre DEVRAIENT seulement utiliser des paramètres de type de support lorsque ils sont exigés par cette définition de type/sous-type.

Les valeurs de type de support sont enregistrées auprès de l'Autorité d'allocation des numéros de l'Internet (IANA [19]). Le processus d'enregistrement du type de support est décrit dans la RFC 1590 [17]. L'utilisation de types de support non enregistrés est déconseillée.

3.7.1 Canonisation et texte par défaut

Les types de support Internet sont enregistrés sous forme canonique. Un corps d'entité transféré via des messages HTTP DOIT être représenté sous la forme canonique appropriée avant sa transmission, sauf pour les types "text", comme défini au paragraphe suivant.

En forme canonique, les sous-types de support de type "text" utilisent le CRLF comme rupture de ligne de texte. HTTP assouplit cette exigence et permet le transport de support texte avec un CR ou LF représentant seul une rupture de ligne lorsque ceci est fait de façon cohérente pour un corps d'entité entier. Les applications HTTP DOIVENT accepter le CRLF, le CR seul, et le LF seul comme représentatifs d'une rupture de ligne en support de texte reçu via HTTP. De plus, si le texte est représenté dans un jeu de caractères qui n'utilise pas les octets 13 et 10 pour respectivement CR et LF, comme c'est le cas pour certains jeux de caractères multi octets, HTTP permet l'utilisation de toute séquence d'octets définie par ce jeu de caractères pour représenter les équivalents du CR et du LF pour les coupures de ligne. Cette souplesse en ce qui concerne les coupures de ligne ne s'applique qu'au support de texte dans le corps d'entité ; un CR seul ou un LF seul NE DOIT PAS être substitué à un CRLF au sein d'une structure de commande HTTP (comme un champ d'en-tête et une frontière de multi-parties).

Si un corps d'entité est codé avec un codage de contenu, les données sous-jacentes DOIVENT être dans une forme définie ci-dessus avant d'être codées.

Le paramètre "charset" est utilisé avec certains types de support pour définir le jeu de caractères (paragraphe 3.4) des données. Lorsque aucun paramètre jeu de caractères explicite n'est fourni par l'envoyeur, les sous-types de support de type "text" sont définis comme ayant la valeur de jeu de caractères par défaut de "ISO-8859-1" lorsqu'ils sont reçus via HTTP. Les données dans des jeux de caractères autres que "ISO-8859-1" ou ses sous ensembles DOIVENT être étiquetées avec une valeur de jeu de caractères appropriée. Voir au paragraphe 3.4.1 pour les problèmes de compatibilité.

3.7.2 Types multi parties

MIME fournit un certain nombre de types "multipart" – qui encapsulent une ou plusieurs entités au sein d'un seul corps de message. Tous les types multi parties partagent une syntaxe commune, comme défini au paragraphe 5.1.1 de la RFC 2046 [40], et DOIVENT inclure un paramètre de limite *(boundary)* au titre de la valeur de type de support. Le corps de message est lui-même un élément de protocole et DOIT donc n'utiliser que le CRLF pour représenter les coupures de ligne entre les parties de corps. À la différence de la RFC 2046, l'épilogue de tout message multi parties DOIT être vide ; les applications HTTP NE DOIVENT PAS transmettre l'épilogue (même si le message multi parties original contient un épilogue). Ces restrictions existent afin de préserver la nature auto-délimitante d'un corps de message multi parties, où la "fin" du corps de message est indiquée par la fin de la limite de multi parties.

En général, HTTP ne traite pas un corps de message multi parties différemment d'aucun autre type de support : strictement comme une charge utile. La seule exception est le type "multipart/byteranges" (appendice 19.2) lorsqu'il apparaît dans une réponse 206 (Contenu partiel) qui sera interprété par certains mécanismes de mise en antémémoire HTTP comme décrit aux paragraphes 13.5.4 et 14.16. Dans tous les autres cas, un agent d'utilisateur HTTP DEVRAIT suivre le même comportement, ou un comportement similaire à celui qu'aurait un agent d'utilisateur MIME à réception d'un type multi parties. Les champs d'en-tête MIME au sein de chaque partie de corps d'un corps de message multi parties n'ont aucune signification pour HTTP au delà de ce qui est défini par leur sémantique MIME.

En général, un agent d'utilisateur HTTP DEVRAIT suivre le même comportement, ou un comportement similaire à celui qu'aurait un agent d'utilisateur MIME à réception d'un type multi parties. Si une application reçoit un sous-type multi parties non reconnu, l'application DOIT le traiter comme si il était équivalent à "multipart/mixed".

Note : Le type "multipart/form-data" a été spécifiquement défini pour porter des données de forme convenable pour un traitement via la méthode de demande POST, comme décrit à la RFC 1867 [15].

3.8 Jetons product

Les jetons product *(produit)* sont utilisés pour permettre à des applications communicantes de s'identifier par le nom et la version de logiciel. La plupart des champs qui utilisent les jetons produits permettent aussi des sous-produits qui forment une partie significative de l'application à lister, séparés par une espace blanche. Par convention, la liste des produits est faite dans l'ordre de leur signification pour l'identification de l'application.

```
product  = token ["/" product-version]
product-version   = token
```

Exemples :

```
User-Agent: CERN-LineMode/2.15 libwww/2.17b3
Server: Apache/0.8.4
```

Les jetons Product DEVRAIENT être courts et concis. Ils NE DOIVENT PAS être utilisés pour de la publicité ou autre information non essentielle. Bien que tout caractère de jeton PUISSE apparaître dans une version de produit, ce jeton DEVRAIT seulement être utilisé pour un identifiant de version (c'est-à-dire que des versions successives du même produit DEVRAIENT ne différer que par la portion version de produit de la valeur de produit).

3.9 Valeurs de qualité

La négociation de contenu HTTP (section 12) utilise de courts nombres "à virgule flottante" pour indiquer l'importance relative ("le poids") des divers paramètres négociables. Un poids est normalisé par un nombre réel dans la gamme de 0 à 1, où 0 est la valeur minimum et 1 la valeur maximum. Si un paramètre a une valeur de

qualité de 0, le contenu ayant ce paramètre est 'non acceptable' pour le client. Les applications HTTP/1.1 NE DOIVENT PAS générer plus de trois chiffres après la virgule. La configuration d'usager de ces valeurs DEVRAIT aussi être limitée de cette façon.

```
qvalue  = ( "0" [ "." 0*3DIGIT ] ) | ( "1" [ "." 0*3("0") ] )
```

"Valeur de qualité" est un nom mal approprié, car ces valeurs représentent seulement une dégradation relative de la qualité désirée.

3.10 Étiquettes de langage

Une étiquette de langage identifie un langage naturel parlé, écrit, ou autrement porté par des êtres humains pour la communication d'informations avec d'autres êtres humains. Les langages d'ordinateurs sont explicitement exclus. HTTP utilise des étiquettes *(tags)* de langage dans les champs Accept-Language *(langue acceptée)* et Content-Language *(langue du contenu)*.

La syntaxe et l'enregistrement des étiquettes de langage HTTP sont les mêmes que celles définies par la RFC 1766 [1]. En résumé, une étiquette de langage se compose d'une ou plusieurs parties : une étiquette de langage principal et une série de sous-étiquettes éventuellement vides :

```
language-tag  = primary-tag *( "-" subtag )
primary-tag  = 1*8ALPHA
subtag  = 1*8ALPHA
```

L'espace blanche n'est pas admise au sein de l'étiquette et toutes les étiquettes sont insensibles à la casse. L'espace de nom des étiquettes de langage est administré par l'IANA. Des exemple d'étiquette de langage sont : en, en-US, en-cockney, i-cherokee, x-pig-latin, où les deux premières lettres de l'étiquette sont une abréviation ISO-639 de la langue et les deux lettres initiales de la sous-étiquette sont le code de pays ISO-3166. (Les trois dernières étiquettes ci-dessus ne sont pas des étiquettes enregistrées ; toutes sauf la dernière sont des exemples d'étiquettes qui pourraient être enregistrées à l'avenir.)

3.11 Étiquettes d'entité

Les étiquettes d'entité sont utilisées pour comparer deux ou plusieurs entités provenant de la même ressource demandée. HTTP/1.1 utilise les étiquettes d'entité dans les champs d'en-tête ETag (paragraphe 14.19), If-Match (paragraphe 14.24), If-None-Match (paragraphe 14.26), et If-Range (paragraphe 14.27). La définition de la façon dont elles sont utilisées et comparées comme valideurs d'antémémoire figure au paragraphe 13.3.3. Une étiquette d'entité consiste en une chaîne opaque entre guillemets, avec un éventuel préfixe d'indicateur de faiblesse.

```
entity-tag   = [ weak ] opaque-tag
weak         = "W/"
opaque-tag   = quoted-string
```

Une "étiquette d'entité forte" ne PEUT être partagée par deux entités d'une ressource que si elles sont équivalentes par égalité d'octet.

Une "étiquette d'entité faible" indiquée par le préfixe "W/", ne PEUT être partagée par deux entités d'une ressource que si elles sont équivalentes et pourraient être substituées l'une à l'autre sans changement significatif de la sémantique. Une étiquette d'entité faible peut seulement être utilisée pour une comparaison faible.

Une étiquette d'entité DOIT être unique à travers toutes les versions de toutes les entités associées à une ressource particulière. Une valeur d'étiquette d'entité donnée PEUT être utilisée pour des entités obtenues par des demandes sur des URI différents. L'utilisation de la même valeur d'étiquette d'entité conjointement avec des entités obtenues par des demandes sur des URI différents n'implique pas l'équivalence de ces entités.

3.12 Unités de gamme

HTTP/1.1 permet à un client de demander que seule une partie (une gamme de) l'entité de réponse soit incluse

dans la réponse. HTTP/1.1 utilise les unités de gamme dans les champs d'en-tête Range (paragraphe 14.35) et Content-Range (paragraphe 14.16). Une entité peut être éclatée en sous-gammes selon diverses unités structurelles.

```
range-unit         = bytes-unit | other-range-unit
bytes-unit         = "bytes"
other-range-unit   = token
```

La seule unité de gamme définie par HTTP/1.1 est "bytes". Les mises en œuvre de HTTP/1.1 PEUVENT ignorer les gammes spécifiées en utilisant d'autres unités.

HTTP/1.1 a été conçu pour permettre les mises en œuvre d'applications qui ne dépendent pas de la connaissance des gammes.

4 Message HTTP

4.1 Types de message

Les messages HTTP consistent en demandes du client au serveur et en réponses du serveur au client.

```
HTTP-message  = Request | Response  ; messages HTTP/1.1
```

Les messages Request (section 5) et Response (section 6) utilisent le format de message générique de la RFC 822 [9] pour le transfert des entités (la charge utile du message). Les deux types de message consistent en une ligne de début, zéro, un ou plusieurs champs d'en-tête (aussi appelés "header"), une ligne vide (c'est-à-dire, une ligne sans rien avant le CRLF) indiquant la fin des champs d'en-tête, et éventuellement un corps de message.

```
message-générique = ligne-de-début
*(CRLF d'en-tête de message)
CRLF
[ corps de message ]
start-line  = Request-Line | Status-Line
```

Dans l'intérêt de la robustesse, les serveurs DEVRAIENT ignorer toute ligne vide reçue lorsqu'on attend une Request-Line. En d'autres termes, si le serveur est en train de lire le flux de protocole au début d'un message et qu'il reçoit d'abord un CRLF, il devrait ignorer le CRLF.

Certaines mises en œuvre défectueuses de client HTTP/1.0 génèrent des CRLF excédentaires après une demande POST. Pour répéter ce qui était explicitement interdit par le BNF, un client HTTP/1.1 NE DOIT PAS préfacer ou faire suivre une demande d'un CRLF supplémentaire

4.2 En-têtes de message

Les champs d'en-tête HTTP, qui incluent les champs general-header (paragraphe 4.5), request-header (paragraphe 5.3), response-header (paragraphe 6.2), et entity-header (paragraphe 7.1), suivent le même format général que celui donné au paragraphe 3.1 de la RFC 822 [9]. Chaque champ d'en-tête consiste en un nom suivi par deux points (":") et le champ valeur. Les noms de champ sont insensibles à la casse. La valeur de champ PEUT être précédée par un nombre quelconque de LWS, bien qu'un seul SP soit préféré. Les champs d'en-tête peuvent être étendus sur plusieurs lignes en faisant précéder chaque ligne supplémentaire par au moins un SP ou HT. Les applications devraient suivre la "forme commune", lorsque une est connue ou indiquée, lors de la génération de constructions HTTP, car il peut exister quelques mises en œuvre qui ne réussissent pas à accepter tout ce qui n'est pas une forme commune.

```
en-tête-de-message = nom-de-champ ":" [ valeur-de-champ ]
nom-de-champ       = jeton
valeur-de-champ    = *( contenu-de-champ | LWS )
contenu-de-champ   = <les OCTETS qui constituent la valeur de champ et consistent en
                   *TEXT ou combinaisons de jetons, séparateurs, et chaînes entre
                   guillemets>
```

Le contenu-de-champ n'inclut aucun LWS en tête ou en finale : les espaces linéaires qui surviennent avant le

premier caractère non espace de la valeur de champ ou après le dernier caractère non espace de la valeur de champ. Un tel LWS de tête ou de finale PEUT être retiré sans changer la sémantique de la valeur de champ. Tout LWS qui survient au sein du contenu de champ PEUT être remplacé par un seul SP avant d'interpréter la valeur de champ ou de transmettre le message vers l'aval.

L'ordre dans lequel les champs d'en-tête avec des noms de champ différents sont reçus n'est pas significatif. Cependant, il est de "bonne pratique" d'envoyer d'abord les champs d'en-tête généraux, suivis par les champs d'en-tête d'en-tête de demande ou de réponse, et de terminer par des champs d'en-tête d'entité.

Plusieurs champs d'en-tête de message avec le même nom de champ PEUVENT être présents dans un message si et seulement si la valeur de champ entière pour ce champ d'en-tête est définie comme une liste à octets séparés par des virgules [c'est-à-dire, différentes (valeurs)]. Il DOIT être possible de combiner les multiples champs d'en-tête en une paire "nom-de-champ: valeur-de-champ", sans changer la sémantique du message, en ajoutant chaque valeur de champ à la première, chacune séparée par une virgule. L'ordre dans lequel les champs d'en-tête avec le même nom-de-champ sont reçus est donc significatif pour l'interprétation de la valeur de champ combinée, et donc, un mandataire NE DOIT PAS changer l'ordre de ces valeurs de champ lors de la transmission d'un message.

4.3 Corps de message

Le corps de message (s'il en est) d'un message HTTP est utilisé pour transporter le corps d'entité associé à la demande ou réponse. Le corps de message diffère seulement du corps d'entité lorsque un codage de transfert a été appliqué, comme indiqué par le champ d'en-tête Transfer-Encoding (paragraphe 14.41).

```
corps de message  = corps d'entité  | <corps d'entité codé selon Transfer-Encoding>
```

Transfer-Encoding DOIT être utilisé pour indiquer tous codages de transfert appliqués par une application pour assurer un transfert sûr et approprié du message. Transfer-Encoding est une propriété du message, et non de l'entité, et donc PEUT être ajouté ou retiré par toute application le long de la chaîne des demandes/réponses. (Cependant, le paragraphe 3.6 fait des restrictions quant au moment où certains codages de transfert peuvent être utilisés.)

Les règles sur le moment où un corps de message est admis dans un message diffèrent pour les demandes et les réponses.

La présence d'un corps de message dans une demande est signalée par l'inclusion d'un champ d'en-tête Content-Length ou Transfer-Encoding dans les en-têtes de message de la demande. Un corps de message NE DOIT PAS être inclus dans une demande si la spécification de la méthode de la demande (paragraphe 5.1.1) ne permet pas d'envoyer un corps d'entité dans des demandes. Un serveur DEVRAIT lire et transmettre un corps de message sur toute demande ; si la méthode de la demande n'inclut aucune sémantique définie pour un corps d'entité, le corps de message DEVRAIT alors être ignoré lors du traitement de la demande.

Pour les messages de réponse, savoir si un corps de message est inclus ou non avec un message dépend à la fois de la méthode de la demande et du code d'état de la réponse (paragraphe 6.1.1). Toutes les réponses à la méthode de demande HEAD NE DOIVENT PAS inclure un corps de message, même si la présence des champs d'en-tête d'entité pourrait conduire à croire qu'elles le font. Toutes les réponses 1xx (information), 204 (pas de contenu), et 304 (non modifié) NE DOIVENT PAS inclure un corps de message. Toutes les autres réponses incluent un corps de message, bien qu'il PUISSE être de longueur zéro.

4.4 Longueur de message

La longueur de transfert d'un message est la longueur du corps de message comme elle apparaît dans le message ; c'est-à-dire, après que tous les codages de transfert ont été appliqués. Lorsque un corps de message est inclus avec un message, la longueur de transfert de ce corps est déterminée par une des propositions suivantes (dans l'ordre) :

1. Tout message de réponse qui "NE DOIT PAS" inclure un corps de message (tels que les réponses 1xx, 204, et 304 et toute réponse à une demande HEAD) est toujours terminé par la première ligne vide après les champs d'en-tête, sans considération des champs d'en-tête d'entité présents dans le message.

2. Si un champ d'en-tête Transfer-Encoding (paragraphe 14.41) est présent et a une valeur autre que "identity", la longueur de transfert est alors définie par l'utilisation du transfert de codage "fragmenté" (paragraphe 3.6), sauf si le message se termine en fermant la connexion.

3. Si un champ d'en-tête Content-Length (paragraphe 14.13) est présent, sa valeur décimale en octets représente à la fois la longueur d'entité et la longueur de transfert. Le champ d'en-tête Content-Length NE DOIT PAS être envoyé si ces deux longueurs sont différentes (c'est-à-dire, si un champ d'en-tête Transfer-Encoding est présent). Si un message est reçu avec à la fois un champ d'en-tête Transfer-Encoding et un champ d'en-tête Content-Length, ce dernier DOIT être ignoré.

4. Si le message utilise le type de support "multipart/byteranges", et si la longueur de transfert n'est pas autrement spécifiée, ce type de support auto délimitant définit la longueur de transfert. Ce type de support NE DOIT PAS être utilisé à moins que l'envoyeur ne sache que le receveur peut l'analyser ; la présence dans une demande d'un en-tête Range avec plusieurs spécificateurs de gamme d'octets de la part d'un client 1.1 implique que le client puisse analyser les réponses multi parties/gamme d'octets.

   Un en-tête de gamme peut être transmis par un mandataire 1.0 qui ne comprend pas les multi parties/gamme d'octets ; dans ce cas, le serveur DOIT délimiter le message en utilisant les méthodes définies aux points 1, 3 ou 5 du présent paragraphe.

5. Par le serveur en fermant la connexion. (Fermer la connexion ne peut être utilisé pour indiquer la fin d'un corps de demande, car cela ne laisserait aucune possibilité au serveur de renvoyer une réponse.)

Pour la compatibilité avec les applications HTTP/1.0, les demandes HTTP/1.1 qui contiennent un corps de message DOIVENT inclure un champ d'en-tête Content-Length valide sauf si le serveur est connu comme étant conforme à HTTP/1.1. Si une demande contient un corps de message et qu'un Content-Length n'est pas donné, le serveur DEVRAIT répondre par 400 (Mauvaise demande) si il ne peut pas déterminer la longueur du message, ou par 411 (longueur exigée) si il souhaite insister pour recevoir un Content-Length valide.

Toutes les applications HTTP/1.1 qui reçoivent des entités DOIVENT accepter le codage de transfert "fragmenté" (paragraphe 3.6), permettant ainsi à ce mécanisme d'être utilisé pour des messages lorsque la longueur du message ne peut pas être déterminée à l'avance.

Les messages NE DOIVENT PAS inclure à la fois un champ d'en-tête Content-Length et un codage de transfert qui ne soit pas d'identité. Si le message inclut un codage de transfert qui ne soit pas d'identité, le Content-Length DOIT être ignoré.

Lorsque un Content-Length est donné dans un message où un corps de message est autorisé, sa valeur de champ DOIT exactement correspondre au nombre des octets dans le corps de message. Les agents d'utilisateur HTTP/1.1 DOIVENT notifier à l'utilisateur qu'une longueur invalide est reçue et détectée.

4.5 Champs d'en-tête généraux

Quelques champs d'en-tête ont une applicabilité générale à la fois pour les messages de demande et de réponse, mais qui ne s'applique pas à l'entité à transférer. Ces champs d'en-tête ne s'appliquent qu'au message à transmettre.

```
general-header = Cache-Control          paragraphe 14.9
               | Connection             paragraphe 14.10
               | Date                   paragraphe 14.18
               | Pragma                 paragraphe 14.32
               | Trailer                paragraphe 14.40
               | Transfer-Encoding      paragraphe 14.41
               | Upgrade                paragraphe 14.42
               | Via                    paragraphe 14.45
               | Warning                paragraphe 14.46
```

Les noms de champ d'en-tête généraux ne peuvent être étendus de façon fiable qu'en combinaison avec un changement de la version de protocole. Cependant, des champs d'en-tête nouveaux ou expérimentaux peuvent recevoir la sémantique des champs d'en-tête généraux si toutes les parties à la communication les

reconnaissent comme étant des champs d'en-tête généraux. Les champs d'en-tête non reconnus sont traités comme des champs d'en-tête d'entité.

5 Request

Un message de demande d'un client à un serveur inclut, au sein de la première ligne de ce message, la méthode à appliquer à la ressource, l'identifiant de la ressource, et la version de protocole utilisée.

```
Request = Request-Line                          paragraphe 5.1
          *(( general-header                    paragraphe 4.5
          | request-header                      paragraphe 5.3
          | entity-header ) CRLF)               paragraphe 7.1
          CRLF
          [ corps de message ]                  paragraphe 4.3
```

5.1 Request-Line

La ligne de demande commence par un jeton de méthode, suivi par l'URI de demande et la version du protocole, et se termine par le CRLF. Les éléments sont séparés par des caractères SP. Aucun CR ou LF n'est permis sauf dans la séquence CRLF finale.

```
Request-Line   = Method SP Request-URI SP HTTP-Version CRLF
```

5.1.1 Method

Le jeton Method indique la méthode à effectuer sur la ressource identifiée par l'URI de demande. La méthode est sensible à la casse.

```
Method = "OPTIONS"              paragraphe 9.2
       | "GET"                  paragraphe 9.3
       | "HEAD"                 paragraphe 9.4
       | "POST"                 paragraphe 9.5
       | "PUT"                  paragraphe 9.6
       | "DELETE"               paragraphe 9.7
       | "TRACE"                paragraphe 9.8
       | "CONNECT"              paragraphe 9.9
       | extension-method
extension-method = jeton
```

La liste des méthodes permises par une ressource peut être spécifiée dans un champ d'en-tête Allow (paragraphe 14.7). Le code de retour de la réponse notifie toujours au client si une méthode est actuellement permise sur une ressource, car l'ensemble des méthodes permises peut changer de façon dynamique. Un serveur d'origine DEVRAIT retourner le code d'état 405 (Méthode non admise) si la méthode est connue par le serveur d'origine mais non admise pour la ressource demandée, et 501 (Non mis en œuvre) si la méthode n'est pas reconnue ou pas mise en œuvre par le serveur d'origine. Les méthodes GET et HEAD DOIVENT être prises en charge par tous les serveurs généralistes. Toute autre méthode est FACULTATIVE ; cependant, si les méthodes ci-dessus sont mises en œuvre, elles DOIVENT être mises en œuvre avec la même sémantique que celle spécifiée à la Section 9.

5.1.2 Request-URI

Request-URI est un Identifiant de ressource uniforme (paragraphe 3.2) et identifie la ressource à laquelle s'applique la demande.

```
Request-URI    = "*" | absoluteURI | abs_path | authority
```

Les quatre options pour Request-URI dépendent de la nature de la demande. L'astérisque "*" signifie que la demande ne s'applique pas à une ressource particulière, mais au serveur lui-même, et n'est admise que

lorsque la méthode utilisée ne s'applique pas nécessairement à une ressource. Un exemple pourrait être

```
OPTIONS * HTTP/1.1
```

La forme d'URI absolue est EXIGÉE lorsque la demande est à faire à un mandataire. Il est demandé au mandataire de transmettre la demande ou de la servir à partir d'une antémémoire valide, et de retourner la réponse. Noter que le mandataire PEUT transmettre la demande à un autre mandataire ou directement au serveur spécifié par l'URI absolu. Afin d'éviter des boucle de demandes, un mandataire DOIT être capable de reconnaître tous ses noms de serveur, y compris tous ses alias, les variations locales, et l'adresse IP numérique. Un exemple de ligne de demande serait :

```
GET http://www.w3.org/pub/WWW/TheProject.html HTTP/1.1
```

Pour permettre la transition en URI absolu dans toutes les demandes des futures versions de HTTP, tous les serveurs HTTP/1.1 DOIVENT accepter la forme d'URI absolu dans les demandes, même si les clients HTTP/1.1 vont seulement les générer dans des demandes à des mandataires.

La forme d'autorité n'est utilisée que par la méthode CONNECT (paragraphe 9.9).

La forme la plus courante de Request-URI est celle utilisée pour identifier une ressource sur un serveur d'origine ou une passerelle. Dans ce cas, le chemin absolu de l'URI DOIT être transmis (voir au paragraphe 3.2.1, abs_path) comme la Request-URI, et la localisation réseau de l'URI (autorité) DOIT être transmise dans un champ d'en-tête Host. Par exemple, un client souhaitant restituer la ressource ci-dessus directement à partir du serveur d'origine créerait une connexion TCP vers le port 80 de l'hôte "www.w3.org" et enverrait les lignes :

```
GET /pub/WWW/TheProject.html HTTP/1.1
Host: www.w3.org
```

suivies du reste de la demande. Noter que le chemin absolu ne peut être vide ; si aucun n'est présent dans l'URI d'origine, il DOIT être donné comme "/" (le serveur racine).

Le Request-URI est transmis dans le format spécifié au paragraphe 3.2.1. Si le Request-URI est codé en utilisant le codage "% HEX HEX" [42], le serveur d'origine DOIT décoder le Request-URI afin d'interpréter correctement la demande. Les serveurs DEVRAIENT répondre aux Request-URI invalides avec un code d'état approprié.

Un mandataire transparent NE DOIT PAS réécrire la partie "abs_path" du Request-URI reçu lorsqu'il le retransmet au prochain serveur, excepté comme noté ci-dessus pour remplacer un abs_path nul par "/".

Note : La règle de "non réécriture" empêche le mandataire de changer la signification de la demande lorsque le serveur d'origine utilise de façon impropre un caractère d'URI non réservé pour un objet réservé. Les développeurs de mises en œuvre devraient être avertis que certains mandataires pré-HTTP/1.1 réécrivaient le Request-URI.

5.2 Ressource identifiée par une demande

La ressource exacte identifiée par une demande Internet est déterminée en examinant à la fois le Request-URI et le champ d'en-tête Host.

Un serveur d'origine qui ne permet pas aux ressources de différer de l'hôte demandé PEUT ignorer la valeur du champ d'en-tête Host lors de la détermination de la ressource identifiée par une demande HTTP/1.1. (Mais voir au paragraphe 19.6.1.1 les autres exigences sur la prise en charge de Host dans HTTP/1.1.)

Un serveur d'origine qui différencie les ressources sur la base de l'hôte demandé (parfois désigné comme hôte virtuel ou vanity) DOIT utiliser les règles suivantes pour déterminer la ressource demandée sur une demande HTTP/1.1 :

1. Si l'URI de demande est un URI absolu, l'hôte fait partie du Request-URI. Toute valeur de champ d'en-tête Host dans la demande DOIT être ignorée.
2. Si l'URI de demande n'est pas un URI absolu, et si la demande inclut un champ d'en-tête Host, l'hôte est déterminé par la valeur du champ d'en-tête Host.

3. Si l'hôte tel que déterminé par la règle 1 ou 2 n'est pas un hôte valide sur le serveur, la réponse DOIT être un message d'erreur 400 (mauvaise demande).

Le receveur d'une demande HTTP/1.0 qui manque d'un champ d'en-tête Host PEUT essayer d'utiliser une méthode heuristique (par exemple, l'examen du chemin d'URI pour trouver quelque chose qui identifie un hôte particulier de façon univoque) afin de déterminer quelle ressource exacte est demandée.

5.3 Champs d'en-tête de demande

Les champs d'en-tête de demande permettent au client de passer des informations supplémentaires sur la demande, et sur le client lui-même, au serveur. Ces champs agissent comme des modificateurs de demande, avec une sémantique équivalente à celle des paramètres sur une invocation de méthode de langage de programmation.

```
request-header = Accept                   paragraphe 14.1
               | Accept-Charset           paragraphe 14.2
               | Accept-Encoding          paragraphe 14.3
               | Accept-Language          paragraphe 14.4
               | Authorization            paragraphe 14.8
               | Expect                   paragraphe 14.20
               | From                     paragraphe 14.22
               | Host                     paragraphe 14.23
               | If-Match                 paragraphe 14.24
               | If-Modified-Since        paragraphe 14.25
               | If-None-Match            paragraphe 14.26
               | If-Range                 paragraphe 14.27
               | If-Unmodified-Since      paragraphe 14.28
               | Max-Forwards            paragraphe 14.31
               | Proxy-Authorization      paragraphe 14.34
               | Range                    paragraphe 14.35
               | Referer                  paragraphe 14.36
               | TE                       paragraphe 14.39
               | User-Agent               paragraphe 14.43
```

Les noms des champs d'en-tête de demande ne peuvent être étendus de façon fiable qu'en combinaison avec un changement de version de protocole. Cependant, des champs d'en-tête nouveaux ou expérimentaux PEUVENT recevoir la sémantique des champs d'en-tête de demande si toutes les parties à la communication les reconnaissent comme des champs d'en-tête. Les champs d'en-tête non reconnus sont traités comme des champs d'en-tête d'entité.

6 Réponse

Après réception et interprétation d'un message de demande, un serveur répond par un message de réponse HTTP.

```
Réponse = Ligne-d'état                          paragraphe 6.1
          *(( en-tête-général                   paragraphe 4.5
          | en-tête-de-réponse                  paragraphe 6.2
          | en-tête-d'entité ) CRLF)            paragraphe 7.1
          CRLF                                  paragraphe 7.2
          [ corps-de-message ]
```

6.1 Status-Line

La première ligne d'un message Response est la ligne d'état, qui comporte la version du protocole suivie par un code d'état numérique et sa phrase de texte associée, avec chaque élément séparé par des caractères SP. Aucun CR ni LF n'est admis excepté dans la séquence de CRLF finale.

```
Ligne-d'état = Version-HTTP SP Code-d'état SP Phrase-de-cause CRLF
```

6.1.1 Code d'état et phrase de cause

L'élément Status-Code est un code de résultat entier de trois chiffres de la tentative de comprendre et satisfaire la demande. Ces codes sont pleinement définis à la Section 10. La phrase de cause est destinée à donner une courte description textuelle du code d'état. Le code d'état est destiné à être utilisé par des automates et la phrase de cause est destinée à l'utilisateur humain. Le client n'est pas obligé d'examiner ou afficher la phrase de cause.

Le premier chiffre du code d'état définit la classe de réponse. Les deux derniers chiffres n'ont aucun rôle de catégorisation. Il y a 5 valeurs pour le premier chiffre.

- 1xx : Information - Demande reçue, poursuite du traitement
- 2xx : Succès – L'action a été bien reçue, comprise et acceptée
- 3xx : Redirection - Des actions ultérieures doivent être entreprises afin de mener la demande à bien
- 4xx : Erreur client – La demande contient une mauvaise syntaxe ou ne peut pas être satisfaite
- 5xx : Erreur serveur - Le serveur a échoué à satisfaire une demande apparemment valide

Les valeurs individuelles des codes d'état numériques définis pour HTTP/1.1, et un ensemble d'exemples des phrases de cause correspondantes sont présentés ci-dessous. Les phrases de cause dont la liste figure ici sont seulement des recommandations -- elles PEUVENT être remplacées par des équivalents locaux sans affecter le protocole.

```
Status-Code =
        "100"  paragraphe 10.1.1 : Continue
        "101"  paragraphe 10.1.2 : Changement de protocole
        "200"  paragraphe 10.2.1 : OK
        "201"  paragraphe 10.2.2 : Créé
        "202"  paragraphe 10.2.3 : Accepté
        "203"  paragraphe 10.2.4 : Informations non impératives
        "204"  paragraphe 10.2.5 : Pas de contenu
        "205"  paragraphe 10.2.6 : Rétablir le contenu
        "206"  paragraphe 10.2.7 : Contenu pariel
        "300"  paragraphe 10.3.1 : Choix multiples
        "301"  paragraphe 10.3.2 : Déplacement définitif
        "302"  paragraphe 10.3.3 : Trouvé
        "303"  paragraphe 10.3.4 : Voir un autre
        "304"  paragraphe 10.3.5 : Non modifié
        "305"  paragraphe 10.3.6 : Utiliser un mandataire
        "307"  paragraphe 10.3.8 : Redirection temporaire
        "400"  paragraphe 10.4.1 : Mauvaise demande
        "401"  paragraphe 10.4.2 : Non autorisé
        "402"  paragraphe 10.4.3 : Paiement exigé
        "403"  paragraphe 10.4.4 : Interdit
        "404"  paragraphe 10.4.5 : Non trouvé
        "405"  paragraphe 10.4.6 : Méthode non admise
        "406"  paragraphe 10.4.7 : Pas acceptable
        "407"  paragraphe 10.4.8 : Authentification du mandataire exigée
        "408"  paragraphe 10.4.9 : Expiration de la demande
        "409"  paragraphe 10.4.10 : Conflit
        "410"  paragraphe 10.4.11 : Parti
        "411"  paragraphe 10.4.12 : Longueur exigée
        "412"  paragraphe 10.4.13 : Echec de précondition
        "413"  paragraphe 10.4.14 : Entité de demande trop grande
        "414"  paragraphe 10.4.15 : URI de demande trop grand
        "415"  paragraphe 10.4.16 : Type de support non pris en charge
        "416"  paragraphe 10.4.17 : Gamme demandée non réalisable
        "417"  paragraphe 10.4.18 : Echec de l'attente
        "500"  paragraphe 10.5.1 : Erreur interne du serveur
        "501"  paragraphe 10.5.2 : Non mis en œuvre
        "502"  paragraphe 10.5.3 : Mauvaise passerelle
        "503"  paragraphe 10.5.4 : Service indisponible
        "504"  paragraphe 10.5.5 : Expiration de la passerelle
        "505"  paragraphe 10.5.6 : Code d'extension de version HTTP non prise en charge

code-d'extension = 3DIGIT
Phrase-de-cause  = *<TEXT, à l'exclusion de CR, LF>
```

Les codes d'état HTTP sont extensibles. Il n'est pas exigé des applications HTTP qu'elles comprennent la signification de tous les codes d'état enregistrés, bien qu'une telle compréhension soit évidemment souhaitable. Cependant, les applications DOIVENT comprendre la classe de tout code d'état, comme indiqué par le premier chiffre, et traiter toute réponse non reconnue comme étant équivalente au code d'état x00 de cette classe, à l'exception qu'une réponse non reconnue NE DOIT PAS être mise en antémémoire. Par exemple, si un code d'état non reconnu de 431 est reçu par le client, il peut en toute sécurité supposer qu'il y a quelque chose qui ne va pas dans sa demande et traiter la réponse comme si il avait reçu un code d'état 400. Dans de tels cas, les agents d'utilisateur DEVRAIENT présenter à l'utilisateur l'entité retournée avec la réponse, car cette entité va vraisemblablement inclure les informations lisibles par l'homme qui expliquent cet état inhabituel.

6.2 Champs d'en-tête de réponse

Les champs d'en-tête de réponse permettent au serveur de passer des informations supplémentaires sur la réponse qui ne peuvent pas être placées dans la ligne d'état. Ces champs d'en-tête donnent des informations sur le serveur et sur l'accès ultérieur à la ressource identifiée par l'URI de demande.

```
en-tête-de-réponse = Accept-Ranges     paragraphe 14.5
                   | Age                paragraphe 14.6
                   | ETag               paragraphe 14.19
                   | Location           paragraphe 14.30
                   | Proxy-Authenticate  paragraphe 14.33
                   | Retry-After        paragraphe 14.37
                   | Server             paragraphe 14.38
                   | Vary               paragraphe 14.44
                   | WWW-Authenticate   paragraphe 14.47
```

Les noms de champ d'en-tête de réponse ne peuvent être étendus qu'en combinaison avec un changement de version de protocole. Cependant, des champs d'en-tête nouveaux ou expérimentaux PEUVENT recevoir la sémantique des champs d'en-tête de réponse si toutes les parties à la communication les reconnaissent comme étant des champs d'en-tête de réponse. Les champs d'en-tête non reconnus sont traités comme des champs d'en-tête d'entité.

7 Entité

Les messages de demande et de réponse PEUVENT transférer une entité si ce n'est pas autrement interdit par la méthode de la demande ou par le code d'état de la réponse. Une entité consiste en des champs d'en-tête d'entité et un corps d'entité, quoique certaines réponses ne comportent que des en-têtes d'entité.

Dans la présente section, l'envoyeur et le receveur se réfèrent tous deux soit au client soit au serveur, selon qui envoie et qui reçoit l'entité.

7.1 Champs d'en-tête d'entité

Les champs d'en-tête d'entité définissent des méta-informations sur le corps d'entité, ou si aucun corps n'est présent, sur la ressource identifiée par la demande. Certaines de ces méta-informations sont FACULTATIVES ; certaines peuvent être EXIGÉES par des parties de la présente spécification.

```
   en-tête-d'entité  = Allow                 paragraphe 14.7
                     | Content-Encoding      paragraphe 14.11
                     | Content-Language      paragraphe 14.12
                     | Content-Length        paragraphe 14.13
                     | Content-Location      paragraphe 14.14
                     | Content-MD5           paragraphe 14.15
                     | Content-Range         paragraphe 14.16
                     | Content-Type          paragraphe 14.17
                     | Expires               paragraphe 14.21
                     | Last-Modified         paragraphe 14.29
                     | en-tête-d'extension

  en-tête-d'extension = en-tête-de-message
```

Le mécanisme d'en-tête d'extension permet à des champs d'en-tête d'entité additionnels d'être définis sans changer le protocole, mais ces champs ne peuvent pas être supposés reconnaissables par le receveur. Les champs d'en-tête non reconnus DEVRAIENT être ignorés par le receveur et DOIVENT être transmis par des mandataires transparents.

7.2 Corps d'entité

Le corps d'entité (s'il en est) envoyé avec une demande ou réponse HTTP est dans un format et un codage définis par les champs d'en-tête d'entité.

```
corps-d'entité     = *OCTET
```

Un corps d'entité n'est présent dans un message que lorsque un corps de message est présent, comme décrit au paragraphe 4.3. Le corps d'entité est obtenu du corps de message en décodant tout codage de transfert qui aurait pu être appliqué pour s'assurer d'un transfert sûr et approprié du message.

7.2.1 Type

Lorsqu'un corps d'entité est inclus avec un message, le type de données de ce corps est déterminé via les champs d'en-tête Content-Type et Content-Encoding. Ceux-ci définissent un modèle de codage ordonné en deux couches :

```
corps-d'entité := Content-Encoding( Content-Type( données ) )
```

Content-Type spécifie le type de support des données sous-jacentes. Content-Encoding peut être utilisé pour indiquer tous les codages de contenu supplémentaires qui sont appliqués aux données, habituellement dans le but de compresser les données, qui sont des propriétés de la ressource demandée. Il n'y a pas de codage par défaut.

Tout message HTTP/1.1 qui contient un corps d'entité DEVRAIT inclure un champ d'en-tête Content-Type définissant le type de support de ce corps. Si et seulement si le type de support n'est pas donné par un champ Content-Type, le receveur PEUT essayer de deviner le type de support par l'inspection de son contenu et/ou le ou les noms d'extension de l'URI utilisé pour identifier la ressource. Si le type de support reste inconnu, le receveur DEVRAIT le traiter comme type "flux d'application/octet".

7.2.2 Longueur d'entité

La longueur d'entité d'un message est la longueur du corps de message avant l'application de tout codage de transfert. Le paragraphe 4.4 définit comment est déterminée la longueur de transfert d'un corps de message.

8 Connexions

8.1 Connexions persistantes

8.1.1 Objet

Avant les connexions persistantes, une connexion TCP séparée a été établie pour s'assortir à chaque URL, augmentant la charge qui pèse sur les serveurs HTTP et causant l'encombrement de l'Internet. L'utilisation d'images en ligne et autres données associées exige souvent qu'un client fasse des demandes multiples sur le même serveur en un bref délai. L'analyse de ces problèmes de performances et les résultats d'un prototype de mise en œuvre sont disponibles [26] [30]. Les expériences et mesures de mise en œuvre réelles sur HTTP/1.1 (RFC 2068) montrent de bons résultats [39]. Des solutions de remplacement ont aussi été explorées, par exemple, T/TCP [27].

Les connexions HTTP persistantes ont un certain nombre d'avantages :

– En ouvrant et fermant moins de connexions TCP, le temps de CPU est économisé dans les routeurs et les hôtes (clients, serveurs, mandataires, passerelles, tunnels, ou antémémoires), et la mémoire utilisée pour les blocs de contrôle de protocole TCP peut être économisée dans les hôtes.

- Des demandes et réponses HTTP peuvent être traitées en parallèle *(pipeline)* sur une connexion. Le traitement en parallèle permet à un client de faire plusieurs demandes sans attendre chacune des réponses, permettant à une seule connexion TCP d'être utilisée beaucoup plus efficacement, avec un temps écoulé plus bref.
- L'encombrement du réseau est réduit en réduisant le nombre de paquets causés par des ouvertures TCP, et en accordant à TCP un temps suffisant pour déterminer l'état d'encombrement du réseau.
- Le délai de latence sur les demandes suivantes est réduit dans la mesure où on ne perd pas de temps dans des procédures de prise de contact d'ouverture de connexion.
- HTTP peut évoluer plus paisiblement, car les erreurs peuvent être rapportées sans qu'on soit pénalisé par la fermeture de la connexion TCP. Les clients qui utiliseront les versions futures de HTTP pourront au mieux essayer de nouvelles caractéristiques, mais si ils communiquent avec un serveur plus ancien, il faudra réessayer avec la vieille sémantique après un rapport d'erreur.

Les mises en œuvre HTTP DEVRAIENT mettre en œuvre des connexions persistantes.

8.1.2 Fonctionnement global

Une différence significative entre HTTP/1.1 et les versions plus anciennes de HTTP est que les connexions persistantes sont le comportement par défaut de toute connexion. C'est-à-dire que, sauf mention contraire, le client DEVRAIT supposer que le serveur va maintenir une connexion persistante, même après des réponses d'erreur de la part du serveur.

Les connexions persistantes fournissent un mécanisme par lequel un client et un serveur peuvent signaler la clôture d'une connexion TCP. Cette signalisation a lieu en utilisant le champ d'en-tête Connexion (paragraphe 14.10). Une fois qu'une clôture a été signalée, le client NE DOIT PAS envoyer d'autres demandes sur cette connexion.

8.1.2.1 Négociation

Un serveur HTTP/1.1 PEUT supposer qu'un client HTTP/1.1 a l'intention de maintenir une connexion persistante jusqu'à ce qu'un en-tête Connexion incluant le jeton de connexion "close" soit envoyé dans la demande. Si le serveur choisit de fermer la connexion immédiatement après l'envoi de la réponse, il DEVRAIT envoyer un en-tête Connexion incluant le jeton de connexion "close".

Un client HTTP/1.1 PEUT s'attendre à ce qu'une connexion reste ouverte, mais devrait décider de la garder ouverte sur la base de la présence d'une réponse en provenance d'un serveur contenant un en-tête Connexion avec le jeton de connexion "close". Dans le cas où le client ne veut pas maintenir une connexion pour plus que cette demande, il DEVRAIT envoyer un en-tête Connexion incluant le jeton de connexion "close".

Si ni le client ni le serveur n'envoient le jeton de fermeture dans l'en-tête Connexion, cette demande devient la dernière de la connexion.

Les clients et serveurs NE DEVRAIENT PAS supposer qu'une connexion persistante est maintenue pour des versions HTTP inférieures à 1.1 sauf explicitement signalé. Voir au paragraphe 19.6.2 des précisions sur la rétro compatibilité avec les clients HTTP/1.0.

Afin de rester persistante, tous les messages sur la connexion DOIVENT avoir une longueur de message auto définie (c'est-à-dire, une longueur qui ne soit pas définie par la clôture de la connexion) comme décrit au paragraphe 4.4.

8.1.2.2 Intubation

Un client qui prend en charge les connexions persistantes PEUT "traiter en parallèle" ses demandes (c'est-à-dire, envoyer plusieurs demandes sans attendre chacune des réponses). Un serveur DOIT envoyer ses réponses à ces demandes dans le même ordre que celui dans lequel les demandes ont été reçues.

Les clients qui supposent des connexions persistantes et traitent en parallèle immédiatement après l'établissement de la connexion DEVRAIENT être prêts à réessayer leur connexion si la première tentative de taitement en parallèle échoue. Si un client fait un tel nouvel essai, il NE DOIT PAS traiter en parallèle avant qu'il sache que la connexion est persistante. Les clients DOIVENT aussi être prêts à renvoyer leurs demandes si le

serveur ferme la connexion avant d'envoyer toutes les réponses correspondantes.

Les clients NE DEVRAIENT PAS traiter en parallèle des demandes en utilisant des méthodes non idempotentes ou des séquences de méthodes non idempotentes (voir au paragraphe 9.1.2). Autrement, une terminaison prématurée de la connexion de transport pourrait conduire à des résultats indéterminés. Un client souhaitant envoyer une demande non idempotente DEVRAIT attendre pour envoyer cette demande jusqu'à ce qu'il ait reçu l'état de réponse pour la demande précédente.

8.1.3 Serveurs mandataires

Il est particulièrement important que les mandataires mettent en œuvre correctement les propriétés du champ d'en-tête Connexion comme spécifié au paragraphe 14.10.

Le serveur mandataire DOIT signaler séparément les connexions persistantes avec ses clients et les serveurs d'origine (ou autre serveurs mandataires) auxquels il se connecte. Chaque connexion persistante s'applique à une seule liaison de transport. Un serveur mandataire NE DOIT PAS établir une connexion HTTP/1.1 persistante avec un client HTTP/1.0 (mais voir à la RFC 2068 [33] des informations et la discussion des problèmes posés par l'en-tête Keep-Alive *(garder en vie)* mis en œuvre par de nombreux clients HTTP/1.0).

8.1.4 Considérations pratiques

Les serveurs vont avoir habituellement une valeur de temporisation au-delà de laquelle ils ne vont plus maintenir une connexion inactive. Les serveurs mandataires peuvent augmenter cette valeur car il est vraisemblable que le client va établir d'autres connexions à travers le même serveur. L'utilisation de connexions persistantes ne fait peser aucune exigence quant à la longueur (ou l'existence) de cette temporisation ni sur le client ni sur le serveur.

Lorsqu'un client ou serveur souhaite mettre fin à une temporisation, il DEVRAIT produire une terminaison volontaire sur la connexion de transport. Les clients et serveurs DEVRAIENT à la fois surveiller de façon constante l'autre côté de la fermeture du transport, et y répondre de façon appropriée. Si un client ou serveur ne détecte pas rapidement la fermeture de l'autre côté, il peut causer une perte inutile de ressource sur le réseau.

Un client, serveur, ou mandataire PEUT fermer la connexion de transport à tout moment. Par exemple, un client pourrait avoir commencé d'envoyer une nouvelle demande au moment même où le serveur décide de fermer la connexion "inactive". Du point de vue du serveur, la connexion doit être fermée parce qu'elle était inactive, mais du point de vue du client, une demande est en cours.

Cela signifie que les clients, serveurs, et mandataires DOIVENT être capables de récupérer à partir d'événements de clôture asynchrones. Les logiciels clients DEVRAIENT rouvrir la connexion de transport et retransmettre la séquence interrompue des demandes sans interaction avec l'utilisateur tant que la séquence de demandes est idempotente (voir au paragraphe 9.1.2). Les méthodes ou séquences non idempotentes NE DOIVENT PAS être réessayées automatiquement, bien que les agents d'utilisateur PUISSENT offrir à un opérateur humain le choix de réessayer la ou les demandes. La confirmation par un logiciel d'agent utilisateur avec une compréhension de la sémantique de l'application PEUT se substituer à la confirmation par l'utilisateur. Le réessai automatique NE DEVRAIT PAS être répété si la seconde séquence des demandes échoue.

Les serveurs DEVRAIENT toujours répondre à au moins une demande par connexion, si c'est possible. Les serveurs NE DEVRAIENT PAS fermer une connexion au milieu de la transmission d'une réponse, sauf si on soupçonne un échec du réseau ou du client.

Les clients qui utilisent des connexions permanentes DEVRAIENT limiter le nombre de connexions simultanées qu'ils maintiennent sur un serveur donné. Un client d'un seul utilisateur NE DEVRAIT PAS maintenir plus de deux connexions avec tout serveur ou mandataire. Un mandataire DEVRAIT utiliser jusqu'à 2*N connexions avec un autre serveur ou mandataire, où N est le nombre d'utilisateurs actifs simultanés. Ces lignes directrices sont destinées à améliorer les temps de réponse de HTTP et éviter les encombrements.

8.2 Exigences pour la transmission de message

8.2.1 Connexions persistantes et contrôle de flux

Les serveurs HTTP/1.1 DEVRAIENT maintenir des connexions persistantes et utiliser les mécanismes de contrôle de flux de TCP pour résoudre les surcharges temporaires, plutôt que de mettre fin aux connexions en espérant que les clients vont réessayer. Cette dernière technique peut exacerber l'encombrement du réseau.

8.2.2 Surveillance des connexions pour les messages d'état d'erreur

Un client HTTP/1.1 (ou d'une version plus récente) qui envoie un corps de message DEVRAIT surveiller les états d'erreur sur la connexion réseau lorsqu'il est en train d'envoyer la demande. Si le client voit un état d'erreur, il DEVRAIT immédiatement cesser la transmission du corps de message. Si le corps de message est à envoyer en utilisant un codage "fragmenté" (paragraphe 3.6) un fragment de longueur zéro et une terminaison vide PEUVENT être utilisés pour marquer prématurément la fin du message. Si le corps du message était précédé d'un en-tête Longueur-de-contenu, le client DOIT fermer la connexion.

8.2.3 Utilisation de l'état 100 (Continue)

L'objet de l'état 100 (Continue) (voir au paragraphe 10.1.1) est de permettre au client qui envoie un message de demande avec un corps de demande de déterminer si le serveur d'origine est d'accord pour accepter la demande (sur la base des en-têtes de la demande) avant que le client n'envoie le corps de la demande. Dans certains cas, il peut être inapproprié ou très inefficace pour le client d'envoyer le corps du message si le serveur va rejeter le message sans en regarder le corps.

Exigences pour les clients HTTP/1.1 :

- Si un client veut attendre une réponse 100 (Continue) avant d'envoyer le corps de la demande, il DOIT envoyer un champ d'en-tête de demande Expect (paragraphe 14.20) avec l'attente de "100-continue".
- Un client NE DOIT PAS envoyer un champ d'en-tête de demande Expect (paragraphe 14.20) avec l'attente de "100-continue" si il n'a pas l'intention d'envoyer un corps de demande.

À cause de la présence de mises en œuvre plus anciennes, le protocole permet des situations ambiguës dans lesquelles un client peut envoyer "Expect: 100- continue" sans recevoir ni un état 417 (échec d'attente) ni un état 100 (Continue). Donc, lorsqu'un client envoie ce champ d'en-tête à un serveur d'origine (éventuellement via un mandataire) duquel il n'a jamais reçu un état 100 (Continue), le client NE DEVRAIT PAS attendre pendant une période indéfinie avant d'envoyer le corps de la demande.

Exigences pour les serveurs d'origine HTTP/1.1 :

À réception d'une demande qui comporte un champ d'en-tête de demande Expect avec l'attente "100-continue", un serveur d'origine DOIT soit répondre par un état 100 (Continue) et continuer à lire le flux d'entrée, soit répondre par un code d'état final. Le serveur d'origine NE DOIT PAS attendre le corps de la demande avant d'envoyer la réponse 100 (Continue). Si il répond par un code d'état final, il PEUT clore la connexion de transport ou il PEUT continuer à lire et supprimer le reste de la demande. Il NE DOIT PAS effectuer la méthode demandée si il retourne un code d'état final.

Un serveur d'origine NE DEVRAIT PAS envoyer une réponse 100 (Continue) si le message de demande n'inclut pas un champ d'en-tête de demande Expect avec l'attente "100-continue", et NE DOIT PAS envoyer une réponse 100 (Continue) si une telle demande vient d'un client HTTP/1.0 (ou d'une version plus ancienne). Par exception à cette règle, pour la compatibilité avec la RFC 2068, un serveur PEUT envoyer un état 100 (Continue) en réponse à une demande HTTP/1.1 PUT ou POST qui n'inclut pas de champ d'en-tête de demande Expect avec l'attente "100-continue". Cette exception, dont l'objet est de minimiser les délais de traitement de client associés à une attente non déclarée d'un état 100 (Continue) ne s'applique qu'aux demandes HTTP/1.1, et pas à des demandes avec d'autres valeurs de version HTTP.

Un serveur d'origine PEUT omettre une réponse 100 (Continue) si il a déjà reçu tout ou partie du corps de la demande pour la demande correspondante.

Un serveur d'origine qui envoie une réponse 100 (Continue) DOIT en fin de compte envoyer un code d'état final, une fois que le corps de la demande est reçu et traité, à moins qu'il ne termine prématurément la connexion de transport.

Si un serveur d'origine reçoit une demande qui ne comporte pas de champ d'en-tête de demande Expect avec l'attente "100-continue", la demande inclut un corps de demande, et le serveur répond par un code d'état final avant de lire le corps de demande entier de la part de la connexion de transport, le serveur NE DEVRAIT PAS alors clore la connexion de transport tant qu'il n'a pas lu la demande entière, ou jusqu'à ce que le client close la connexion. Autrement, le client pourrait ne pas recevoir de façon fiable le message de réponse. Cependant, cette exigence n'est pas à interpréter comme empêchant un serveur de se défendre contre des attaques de déni de service, ou contre des mises en œuvre client gravement endommagées.

Exigences pour les mandataires HTTP/1.1 :

– Si un mandataire reçoit une demande qui inclut un champ d'en-tête de demande Expect avec l'attente "100-continue", et si le mandataire sait que le serveur du prochain bond est conforme à HTTP/1.1 ou à une version plus récente, ou s'il ne connaît pas la version HTTP du serveur du prochain bond, il DOIT transmettre la demande, y compris le champ d'en-tête Expect.

– Si le mandataire sait que la version du serveur du prochain bond est HTTP/1.0 ou une version plus ancienne, il NE DOIT PAS transmettre la demande, et il DOIT répondre par un état 417 (Échec de l'attente).

– Les mandataires DEVRAIENT maintenir un enregistrement d'antémémoire des numéros de version HTTP reçus des serveurs de prochain bond récemment référencés.

– Un mandataire NE DOIT PAS transmettre une réponse 100 (Continue) si le message de demande a été reçu d'un client HTTP/1.0 (ou une version plus ancienne) et n'incluait pas de champ d'en-tête de demande Expect avec l'attente "100-continue". Cette exigence prend le pas sur la règle générale de transmission des réponses 1xx (paragraphe 10.1).

8.2.4 Comportement du client si le serveur clôture la connexion de façon prématurée

Si un client HTTP/1.1 envoie une demande qui comporte un corps de demande, mais ne comporte pas de champ d'en-tête de demande Expect avec l'attente "100-continue", et si le client n'est pas directement connecté à un serveur d'origine HTTP/1.1, et si le client voit fermer la connexion avant d'avoir reçu aucun état de la part du serveur, le client DEVRAIT réessayer la demande. Si le client ne réessaye pas cette demande, il PEUT utiliser l'algorithme "attente exponentielle binaire" suivant pour être assuré d'obtenir une réponse fiable :

1. Initier une nouvelle connexion avec le serveur.
2. Transmettre les en-têtes de la demande.
3. Initier un R variable pour le délai d'aller-retour estimé jusqu'au serveur (par exemple, sur la base du temps qu'il lui a fallu pour établir la connexion), ou pour une valeur constante de 5 secondes si le délai d'aller retour n'est pas disponible.
4. Calculer T = R * (2**N), où N est le nombre d'essais précédents de cette demande.
5. Attendre une réponse d'erreur de la part du serveur, ou pendant T secondes (selon celui qui survient le premier).
6. Si aucune réponse d'erreur n'est reçue, transmettre le corps de la demande après T secondes.
7. Si le client voit que la connexion est close prématurément, répéter à partir de l'étape 1 jusqu'à ce que la demande soit acceptée, qu'une réponse d'erreur soit reçue, ou que l'utilisateur devienne impatient et termine le processus d'essai.

Si un état d'erreur est reçu à un moment quelconque, le client

– NE DEVRAIT PAS continuer et
– DEVRAIT clore la connexion si il n'a pas achevé l'envoi du message de demande.

9 Définitions des méthodes

L'ensemble des méthodes communes pour HTTP/1.1 est défini ci-dessous. Bien que cet ensemble puisse être étendu, des méthodes supplémentaires ne peuvent être supposées partager la même sémantique pour des clients et serveurs ayant des extensions distinctes.

Le champ d'en-tête de demande Host (paragraphe 14.23) DOIT accompagner toutes les demandes HTTP/1.1.

9.1 Méthodes sûres et idempotentes

9.1.1 Méthodes sûres

Les développeurs devraient avoir conscience que le logiciel représente l'utilisateur dans ses interactions sur l'Internet, et ils devraient veiller à permettre à l'usager d'être au courant de toute action qui pourrait avoir une signification inattendue pour lui-même ou pour les autres.

En particulier, il a été établi une convention par laquelle les méthodes GET et HEAD NE DEVRAIENT PAS signifier d'entreprendre une autre action que la restauration. Ces méthodes devraient être considérées comme "sûres". Ceci permet aux agents d'utilisateur de représenter d'autres méthodes, telles que POST, PUT et DELETE, d'une façon particulière, de sorte que l'usager soit au courant du fait qu'il est possible qu'une action non sûre soit demandée.

Naturellement, il n'est pas possible de garantir que le serveur ne génère pas des effets collatéraux suite à l'exécution d'une demande GET ; en fait, certaines ressources dynamiques considèrent cela comme une caractéristique. La distinction important ici est que l'usager n'a pas demandé les effets collatéraux, et ne peut donc en être tenu pour responsable.

9.1.2 Méthodes idempotentes

Les méthodes peuvent aussi avoir la propriété d'"idempotence" en ce que (à part les questions d'erreur ou d'expiration) les effets collatéraux de N > 0 demandes sont les mêmes que pour une seule demande. Les méthodes GET, HEAD, PUT et DELETE partagent cette propriété. Aussi, les méthodes OPTIONS et TRACE NE DEVRAIENT PAS avoir d'effets collatéraux, et sont dont par nature idempotentes.

Cependant, il est possible qu'une séquence de plusieurs demandes soit non idempotente, même si toutes les méthodes exécutées dans cette séquence sont idempotentes. (Une séquence est idempotente si une seule exécution de la séquence entière donne toujours un résultat qui n'est pas changé par une réexécution de tout, ou partie, de cette séquence.) Par exemple, une séquence est non idempotente si son résultat dépend d'une valeur qui est modifiée ultérieurement dans la même séquence.

Une séquence qui n'a jamais d'effets collatéraux est idempotente, par définition (pourvu qu'aucune opération concurrente ne soit exécutée sur le même ensemble de ressources).

9.2 OPTIONS

La méthode OPTIONS représente une demande d'informations sur les options de communication disponibles sur la chaîne des demandes/réponses identifiées par l'URI de demande. Cette méthode permet au client de déterminer les options et/ou exigences associées à une ressource, ou les capacités d'un serveur, sans impliquer une action de ressource ou sans initier de restitution de ressource.

Les réponses à cette méthode ne peuvent pas être mises en antémémoire.

Si la demande OPTIONS inclut un corps d'entité (comme indiqué par la présence de Content-Length ou Transfer-Encoding) le type de support DOIT être indiqué par un champ Content-Type. Bien que la présente spécification ne définisse pas l'utilisation d'un tel corps, des extensions futures à HTTP pourraient utiliser les corps OPTIONS pour faire des questions plus précises au serveur. Un serveur qui ne prend pas en charge une telle extension PEUT éliminer le corps de la demande.

Si l'URI de demande est un astérisque ("*") la demande OPTIONS est destinée à s'appliquer au serveur en général plutôt qu'à une ressource spécifique. Comme les options de communication d'un serveur dépendent normalement de la ressource, la demande "*" n'est utile que comme type de méthode "ping" ou "no-op" ; elle ne fait rien de plus que permettre au client de tester les capacités du serveur. Par exemple, cela peut être utilisé pour tester la conformité d'un mandataire à HTTP/1.1 (ou son manque de conformité).

Si l'URI de demande n'est pas un astérisque, la demande OPTIONS ne s'applique qu'aux options qui sont disponibles lors de la communication avec cette ressource.

Une réponse 200 DEVRAIT inclure tous les champs d'en-tête qui indiquent des caractéristiques facultatives mises en œuvre par le serveur et applicables à cette ressource (par exemple, Allow), incluant éventuellement des extensions non définies par la présente spécification. Le corps de réponse, s'il en est, DEVRAIT aussi inclure des informations sur les options de communication. Le format d'un tel corps n'est pas défini par la présente spécification, mais pourrait être défini par des extensions futures à HTTP. La négociation de contenu PEUT être utilisée pour choisir le format de réponse approprié. Si aucun corps de réponse n'est inclus, la réponse DOIT inclure un champ Content-Length d'une valeur de champ de "0".

Le champ d'en-tête de demande Max-Forwards PEUT être utilisé pour cibler un mandataire spécifique dans la chaîne de demande. Lorsqu'un mandataire reçoit une demande OPTIONS sur un URI absolu pour lequel la transmission de la demande est permise, le mandataire DOIT vérifier qu'il y a un champ Max-Forwards. Si la valeur du champ Max-Forwards est zéro ("0"), le mandataire NE DOIT PAS transmettre le message ; au lieu de cela, le mandataire DEVRAIT répondre par ses propres options de communication. Si la valeur du champ Max-Forwards est un entier supérieur à zéro, le mandataire DOIT décrémenter la valeur du champ lorsqu'il transmet la demande. Si aucun champ Max-Forwards n'est présent dans la demande, la demande transmise NE DOIT PAS alors inclure de champ Max-Forwards.

9.3 GET

La méthode GET signifie la restitution de toute information (sous forme d'une entité) qui est identifiée par l'URI de demande. Si l'URI de demande se réfère à un processus de production de données, ce sont les données produites qui devront être retournées comme entité dans la réponse et non le texte source du processus, sauf si ce texte se trouve être le résultat du processus.

La sémantique de la méthode GET se change en un "GET conditionnel" si le message de demande inclut un champ d'en-tête If-Modified-Since, If-Unmodified-Since, If-Match, If-None-Match, ou If-Range. Une méthode GET conditionnelle demande que l'entité ne soit transférée que dans les circonstances décrites par le ou les champs d'en-tête conditionnels. La méthode GET conditionnelle est destinée à réduire les utilisations non indispensables du réseau en permettant aux entités en antémémoire d'être rafraîchies sans requérir plusieurs demandes ou sans transférer des données déjà détenues par le client.

La sémantique de la méthode GET se change en "GET partiel" si le message de demande inclut un champ d'en-tête Range. Un GET partiel demande que seule une partie de l'entité soit transférée, comme décrit au paragraphe 14.35. La méthode GET partiel est destinée à réduire l'utilisation non indispensable du réseau en permettant que les entités partiellement restituées soient complétées sans transférer des données déjà détenues par le client.

La réponse à une demande GET peut être mise en antémémoire si et seulement si elle satisfait aux exigences de mise en antémémoire pour HTTP décrites dans la section 13.

Voir au paragraphe 15.1.3 les considérations de sécurité lorsque elle est utilisée pour les formes.

9.4 HEAD

La méthode HEAD est identique à GET sauf que le serveur NE DOIT PAS retourner un corps de message dans la réponse. Les méta-informations contenues dans les en-têtes HTTP en réponse à une demande HEAD DEVRAIENT être identiques aux informations envoyées en réponse à une demande GET. Cette méthode peut être utilisée pour obtenir des méta-informations sur l'entité impliquée par la demande sans transférer le corps de l'entité lui-même. Cette méthode est souvent utilisée pour tester la validité, l'accessibilité et les modifications récentes, des liens hypertextes.

La réponse à une demande HEAD PEUT être mise en antémémoire dans le sens où les informations contenues dans la réponse PEUVENT être utilisées pour mettre à jour une entité précédemment mise en antémémoire depuis cette ressource. Si la nouvelle valeur de champ indique que l'entité d'antémémoire diffère de l'entité en cours (comme il devrait être indiqué par un changement dans Content-Length, Content-MD5, ETag ou Last-Modified) l'antémémoire DOIT alors traiter l'entrée d'antémémoire comme périmée.

9.5 POST

La méthode POST est utilisée pour demander que le serveur d'origine accepte l'entité incluse dans la demande comme un nouveau subordonné de la ressource identifiée par l'URI de demande dans la ligne de demande. POST est conçu pour permettre à une méthode uniforme de couvrir les fonctions suivantes :

- Annotation des ressources existantes ;
- Postage d'un message dans un bulletin, des nouvelles de groupe, une liste de diffusion, ou groupe d'articles similaires ;
- Fournir un bloc de données, tel que le résultat de la soumission d'un formulaire, à un processus de traitement de données ;
- Étendre une base de données à travers une opération d'ajout.

La fonction réelle effectuée par la méthode POST est déterminée par le serveur et dépend habituellement de l'URI de demande. L'entité postée est subordonnée à cet URI de la même façon qu'un fichier est subordonné à un répertoire qui le contient, qu'un article est subordonné à un groupe de nouvelles auquel il est aposté, ou qu'un enregistrement est subordonné à une base de données.

L'action effectuée par la méthode POST pourrait ne pas résulter en une ressource qui peut être identifiée par un URI. Dans ce cas, 200 (OK) ou 204 (Pas de contenu) est l'état de réponse approprié, selon que la réponse inclut ou non une entité décrivant le résultat.

Si une ressource a été créée sur le serveur d'origine, la réponse DEVRAIT être 201 (Créé) et contenir une entité qui décrive l'état de la demande et se réfère à la nouvelle ressource, et un en-tête de localisation (voir au paragraphe 14.30).

Les réponses à cette méthode ne peuvent pas être mises en antémémoire, sauf si la réponse inclut les champs d'en-tête Cache-Control ou Expires appropriés. Cependant, la réponse 303 (Voir autres) peut être utilisée pour diriger l'agent d'utilisateur sur la restitution d'une ressource mettable en antémémoire.

Les demandes POST DOIVENT obéir aux exigences de transmission de message établies au paragraphe 8.2.

Voir au paragraphe 15.1.3 les considérations sur la sécurité.

9.6 PUT

La méthode PUT demande que l'entité incluse soit mémorisée sous l'URI de demande. Si l'URI de demande se réfère à une ressource déjà existante, l'entité incluse DEVRAIT être considérée comme une version modifiée de celle qui réside sur le serveur d'origine. Si l'URI de demande ne pointe pas sur une ressource existante, et si l'URI est capable d'être défini comme nouvelle ressource par l'agent d'utilisateur demandeur, le serveur d'origine peut créer la ressource avec cet URI. Si une nouvelle ressource est créée, le serveur d'origine DOIT informer l'agent d'utilisateur via la réponse 201 (Créé). Si une ressource existante est modifiée, les codes de réponse 200 (OK) ou 204 (Pas de contenu) DEVRAIENT être envoyés pour indiquer l'achèvement réussi de la demande. Si la ressource n'a pas pu être créée ou modifiée par l'URI de demande, une réponse d'erreur appropriée DEVRAIT être donnée, reflétant la nature du problème. Le receveur de l'entité NE DOIT PAS ignorer un en-tête Content-* (par exemple Content-Range) qu'il ne comprend ou ne met pas en œuvre et DOIT retourner une réponse 501 (Non mis en œuvre) dans de tels cas.

Si la demande passe à travers une antémémoire et si l'URI de demande identifie une ou plusieurs entités actuellement en antémémoire , ces entrées DEVRAIENT être traitées comme périmées. Il n'est pas possible de mettre les réponses à cette méthode en antémémoire.

La différence fondamentale entre les demandes POST et PUT est réfléchie par les significations différentes des URI de demande. L'URI dans une demande POST identifie la ressource qui va traiter l'entité incluse. Cette ressource pourrait être un processus acceptant les données, une passerelle vers un autre protocole, ou une entité distincte qui accepte les annotations. À l'inverse, l'URI dans une demande PUT identifie l'entité incluse dans la demande -- l'agent d'utilisateur sait quel URI est prévu et le serveur NE DOIT PAS essayer d'appliquer la demande à une autre ressource. Si le serveur désire que la demande soit appliquée à un URI différent, il DOIT envoyer une réponse 301 (Déplacement permanent) ; l'agent d'utilisateur PEUT alors prendre sa propre décision pour savoir s'il doit ou non rediriger la demande.

Une seule ressource PEUT être identifiée par de nombreux URI différents. Par exemple, un article peut avoir un

URI pour identifier "la version en cours" qui est différent de l'URI identifiant chaque version particulière. Dans ce cas, une demande PUT sur un URI général pourrait résulter en ce que plusieurs autres URI soient définis par le serveur d'origine.

HTTP/1.1 ne définit pas comment une méthode PUT affecte l'état d'un serveur d'origine.

Les demandes PUT DOIVENT obéir aux exigences de transmission de message établies au paragraphe 8.2.

Sauf spécification contraire pour un en-tête d'entité particulier, les en-têtes d'entité dans la demande PUT DEVRAIENT être appliqués aux ressources créées ou modifiées par le PUT.

9.7 DELETE

La méthode DELETE demande que le serveur d'origine supprime la ressource identifiée par l'URI de demande. Cette méthode PEUT être outrepassée par une intervention humaine (ou autre moyen) sur le serveur d'origine. Le client ne peut pas avoir la garantie que l'opération a été effectuée, même si le code d'état retourné par le serveur d'origine indique que l'action a été menée à bien. Cependant, le serveur NE DEVRAIT indiquer le succès que si, au moment où la réponse est donnée, il a l'intention de supprimer la ressource ou de la déplacer dans une localisation inaccessible.

Une réponse de succès DEVRAIT être 200 (OK) si la réponse inclut une entité décrivant l'état, 202 (Accepté) si l'action n'a pas encore été représentée, ou 204 (Pas de contenu) si l'action a été exécutée mais si la réponse n'incluait pas d'entité.

Si la demande passe à travers une antémémoire et que l'URI de demande identifie une ou plusieurs entités actuellement en antémémoire, ces entrées DEVRAIENT être traitées comme périmées. Les réponses à cette méthode ne sont pas susceptibles d'être mises en antémémoire

9.8 TRACE

La méthode TRACE est utilisée pour invoquer un bouclage distant, de couche application du message de demande. Le receveur final de la demande DEVRAIT refléter le message reçu en retour par le client comme corps d'entité d'une réponse 200 (OK). Le receveur final est le serveur d'origine ou le premier mandataire ou passerelle à recevoir une valeur Max-Forwards de zéro (0) dans la demande (voir au paragraphe 14.31). Une demande TRACE NE DOIT PAS inclure une entité.

TRACE permet au client de voir ce qui va être reçu de l'autre côté de la chaîne de demande et d'utiliser ces données pour tester ou diagnostiquer les informations. La valeur du champ d'en-tête Via (paragraphe 14.45) est d'un intérêt particulier, car elle agit comme une trace de la chaîne de demande. L'utilisation du champ d'en-tête Max-Forwards permet au client de limiter la longueur de la chaîne de demande, ce qui est utile pour tester une chaîne de mandataires qui transmettent des messages dans une boucle infinie.

Si la demande est valide, la réponse DEVRAIT contenir le message de demande entier dans le corps d'entité, avec un Content-Type de "message/http". Les réponses à cette méthode NE DOIVENT PAS être mises en antémémoire.

9.9 CONNECT

La présente spécification réserve le nom de méthode CONNECT à l'utilisation avec un mandataire qui peut passer de façon dynamique à la fonction de tunnel (par exemple, tunnelage SSL [44]).

10 Définitions des codes d'état

Chaque code d'état est décrit ci-dessous, avec une description de la ou des méthodes qu'il peut suivre et toute méta-information exigée dans la réponse.

10.1 Informations 1xx

Cette classe de code d'état indique une réponse provisoire, qui comporte seulement la ligne d'état et les en-têtes facultatifs, et se termine par une ligne vide. Il n'y a pas d'en-tête exigé pour cette classe de codes d'état. Comme HTTP/1.0 n'a pas défini de code d'état 1xx, les serveurs NE DOIVENT PAS envoyer de réponse 1xx à un client HTTP/1.0 sauf dans des conditions expérimentales.

Un client DOIT être prêt à accepter une ou plusieurs réponses d'état 1xx avant une réponse régulière, même si le client n'attend pas de message d'état 100 (Continuer). Les réponses d'état 1xx non attendues PEUVENT être ignorées par un agent d'utilisateur.

Les mandataires DOIVENT transmettre les réponses 1xx, jusqu'à ce que la connexion entre le mandataire et son client ait été close, ou jusqu'à ce que le mandataire lui-même demande la génération de la réponse 1xx. (Par exemple, si un mandataire ajoute un champ "Expect: 100-continue" lorsqu'il transmet une demande, il n'est pas nécessaire de transmettre la ou les réponses 100 (Continuer) correspondantes.)

10.1.1 100 Continue

Le client DEVRAIT continuer suite à cette demande. Cette réponse intérimaire est utilisée pour informer le client que la partie initiale de la demande a été reçue et n'a pas encore été rejetée par le serveur. Le client DEVRAIT continuer en envoyant le reste de la demande ou, si la demande est déjà terminée, ignorer cette réponse. Le serveur DOIT envoyer une réponse finale après la fin de la demande. Voir au paragraphe 8.2.3 un exposé détaillé de l'utilisation et du traitement de ce code d'état.

10.1.2 101 Changement de protocole

Le serveur comprend et accepte de se conformer à la demande du client, via le champ d'en-tête de message Upgrade (paragraphe 14.42), d'un changement du protocole d'application à utiliser sur cette connexion. Le serveur va changer les protocoles au profit de ceux définis par le champ d'en-tête Upgrade de la réponse immédiatement après la ligne vide qui termine la réponse 101.

Le protocole DEVRAIT être changé dans les seuls cas où il est avantageux de le faire. Par exemple, passer à une version plus récente de HTTP est avantageux par rapport aux versions plus anciennes, et passer à un protocole synchrone en temps réel peut être avantageux lors de la livraison de ressources qui utilisent de telles caractéristiques.

10.2 Réussite 2xx

Cette classe de codes d'état indique que la demande du client a été bien reçue, bien comprise et bien acceptée.

10.2.1 200 OK

La demande a réussi. Les informations retournées avec la réponse dépendent de la méthode utilisée dans la demande, par exemple :

GET ; une entité correspondant à la ressource demandée est envoyée dans la réponse ;

HEAD ; des champs d'en-tête d'entité correspondant à la ressource demandée sont envoyés dans la réponse sans aucun corps de message ;

POST ; une entité décrivant ou contenant le résultat de l'action ;

TRACE ; une entité contenant le message de demande tel que reçu par le serveur final.

10.2.2 201 Créé

La demande a été satisfaite et il en résulte qu'une nouvelle ressource a été créée. La ressource nouvellement créée peut être référencée par le ou les URI retournés dans l'entité de la réponse, avec l'URI le plus spécifique

pour la ressource donnée par un champ d'en-tête Location. La réponse DEVRAIT inclure une entité contenant une liste de caractéristiques et localisations de ressources à partir desquelles l'utilisateur ou agent d'utilisateur peut choisir la plus appropriée. Le format de l'entité est spécifié par le type de support donné dans le champ d'en-tête Content-Type. Le serveur d'origine DOIT créer la ressource avant de retourner le code d'état 201. Si l'action ne peut pas être effectuée immédiatement, le serveur DEVRAIT répondre à la place par une réponse 202 (Accepté).

Une réponse 201 PEUT contenir un champ d'en-tête de réponse ETag indiquant la valeur actuelle de l'étiquette de l'entité pour la variante demandée qui vient d'être créée, voir au paragraphe 14.19.

10.2.3 202 Accepté

Le traitement de la demande a été accepté, mais le traitement n'est pas encore achevé. La demande pourra finalement être ou non traitée, comme elle pourrait être non approuvée lorsque le traitement aura réellement lieu. Il n'est pas possible de renvoyer un code d'état à partir d'une opération asynchrone comme celle-ci.

La réponse 202 est intentionnellement non comminatoire. Son objet est de permettre à un serveur d'accepter une demande d'un autre processus (peut- être un processus orienté sur le traitement par lot qui ne fonctionne qu'une seule fois par jour) sans exiger que la connexion au serveur de l'agent d'utilisateur persiste jusqu'à la fin du processus. L'entité retournée avec cette réponse DEVRAIT inclure une indication de l'état en cours de la demande et un pointeur sur un surveillant d'état ou quelque estimation du moment où l'utilisateur peut s'attendre à ce que la demande soit satisfaite.

10.2.4 203 Informations ne se rapportant pas à l'autorisation

Le retour de méta-informations dans l'en-tête d'entité n'est pas l'ensemble définitif tel que disponible d'après le serveur d'origine, mais elles sont rassemblées d'après une copie locale ou de tierce partie. L'ensemble présenté PEUT être un sous-ensemble ou un sur-ensemble de la version originale. Par exemple, inclure des informations d'annotation locale sur la ressource pourrait résulter en un sur-ensemble des méta-informations connues du serveur d'origine. L'utilisation de ce code de réponse n'est pas exigée et n'est appropriée que lorsque la réponse serait autrement 200 (OK).

10.2.5 204 Pas de contenu

Le serveur a satisfait à la demande mais n'a pas besoin de retourner un corps d'entité, et pourrait vouloir retourner des méta-informations mises à jour. La réponse PEUT inclure de méta-informations nouvelles ou mises à jour sous la forme d'en-tête d'entité qui si elles sont présentes DEVRAIENT être associées à la variante demandée.

Si le client est un agent d'utilisateur, il NE DEVRAIT PAS changer sa vision du document par rapport à celle qui a causé l'envoi de la demande. Cette réponse est principalement destinée à permettre l'entrée d'actions qui doivent avoir lieu sans provoquer de changement de la vision du document actif de l'agent d'utilisateur, bien que toutes méta-informations nouvelles ou mises à jour DEVRAIENT être appliquées au document en cours du point de vue actif de l'agent d'utilisateur.

La réponse 204 NE DOIT PAS inclure de corps de message, et donc, elle est toujours terminée par la première ligne vide après les champs d'en-tête.

10.2.6 205 Rétablir le contenu

Le serveur a satisfait à la demande et l'agent d'utilisateur DEVRAIT rétablir la vision du document qui a causé l'envoi de la demande. Cette réponse est principalement destinée à permettre l'entrée d'actions qui doivent avoir lieu via l'entrée de l'utilisateur, suivie par un nettoyage de la forme dans lequel l'entrée est donnée de telle sorte que l'utilisateur puisse facilement initier une autre action d'entrée. La réponse NE DOIT PAS inclure d'entité.

10.2.7 206 Contenu partiel

Le serveur a satisfait à la demande GET partielle pour la ressource. La demande DOIT avoir un champ d'en-tête Range (paragraphe 14.35) indiquant la gamme désirée, et PEUT avoir un champ d'en-tête If-Range (paragraphe 14.27) pour rendre la demande conditionnelle.

La réponse DOIT inclure les champs d'en-tête suivants :
- Un champ d'en-tête Content-Range (paragraphe 14.16) indiquant la gamme incluse dans cette réponse, ou un type de contenu multiparties/gamme d'octets incluant les champs de gamme de contenu pour chaque partie. Si un champ d'en-tête Content-Length est présent dans la réponse, sa valeur DOIT correspondre au nombre réel d'octets transmis dans le corps de message.
- Date.
- ETag et/ou Content-Location, si l'en-tête est à envoyer dans une réponse 200 à la même demande.
- Expires, Cache-Control, et/ou Vary, si la valeur de champ diffère de celle envoyée dans toute réponse précédente à la même variante.

Si la réponse 206 est le résultat d'une demande If-Range qui a utilisé un valideur d'antémémoire fort (voir au paragraphe 13.3.3), la réponse NE DEVRAIT PAS inclure d'autre en-tête d'entité. Si la réponse est le résultat d'une demande If-Range qui a utilisé un valideur faible, la réponse NE DOIT PAS inclure d'autre en-tête d'entité ; ceci est destiné à empêcher les incohérences entre les corps d'entité en antémémoire et les en-têtes mis à jour. Autrement, la réponse DOIT inclure tous les en-têtes d'entité qui auraient été retournés avec une réponse 200 (OK) à la même demande.

Une antémémoire NE DOIT PAS combiner une réponse 206 avec un autre contenu précédemment mis en mémoire si les en-têtes ETag ou Last-Modified ne correspondent pas exactement, voir au paragraphe 13.5.4.

Une antémémoire qui n'accepte pas les en-têtes Range et Content-Range NE DOIT PAS mettre en antémémoire les réponses 206 (Partiel).

10.3 Redirection 3xx

Cette classe de code d'état indique qu'une action supplémentaire doit être entreprise par l'agent d'utilisateur afin de satisfaire la demande. L'action exigée PEUT être effectuée par l'agent d'utilisateur sans interaction avec l'utilisateur si et seulement si la méthode utilisée dans la seconde demande est GET ou HEAD. Un client DEVRAIT détecter les boucles de redirection infinies, car de telles boucles génèrent du trafic réseau à chaque redirection.

Note Les versions précédentes de la présente spécification recommandaient un maximum de cinq redirections. Les développeurs de contenu devraient être conscients qu'il peut y avoir des clients qui mettent en œuvre de telles limites.

10.3.1 300 Choix multiple

La ressource correspond à chacun des éléments d'un ensemble de représentations, chacun ayant sa propre localisation spécifique, et les informations de négociation conduites par l'agent (section 12) sont fournies de telle sorte que l'utilisateur (ou agent d'utilisateur) puisse choisir une représentation préférée et rediriger sa demande sur cette localisation.

Sauf si il s'agit d'une demande HEAD, la réponse DEVRAIT inclure une entité contenant une liste des caractéristiques et localisations de ressource d'après lesquelles l'utilisateur ou agent d'utilisateur peut choisir celle qui est la plus appropriée. Le format d'entité est spécifié par le type de support donné dans le champ d'en-tête Content-Type. Selon le format et les capacités de l'agent d'utilisateur, le choix le plus approprié PEUT être effectué automatiquement. Cependant, la présente spécification ne définit aucune norme pour un tel choix automatique.

Si le serveur a un choix préféré de représentations, il DEVRAIT inclure l'URI spécifique de cette représentation dans le champ Location ; les agents d'utilisateur PEUVENT utiliser la valeur du champ Location pour la redirection automatique. Cette réponse peut être mise en antémémoire sauf mention contraire.

10.3.2 301 Déplacement permanent

La ressource demandée a été allouée à un nouvel URI permanent et toute référence future à cette ressource DEVRAIT utiliser un des URI retournés. Les clients qui possèdent des capacités d'édition de liens devraient relier automatiquement les références à l'URI de demande d'une ou plusieurs des nouvelles références retournées par le serveur, lorsque c'est possible. Cette réponse peut être mise en antémémoire sauf mention contraire.

Le nouvel URI permanent DEVRAIT être donné par le champ Location dans la réponse. Sauf si la méthode de la demande est HEAD, l'entité de la réponse DEVRAIT contenir une brève note hypertexte avec un hyperlien vers le ou les nouveaux URI.

Si le code d'état 301 est reçu en réponse à une demande autre que GET ou HEAD, l'agent d'utilisateur NE DOIT PAS automatiquement rediriger la demande sauf si elle peut être confirmée par l'utilisateur, dans la mesure où cela peut changer les conditions dans lesquelles la demande est produite.

Note : Lors de la redirection automatique d'une demande POST après réception d'un code d'état 301, certains agents d'utilisateur HTTP/1.0 existants vont la changer à tort en une demande GET.

10.3.3 302 Trouvé

La ressource demandée réside temporairement sous un URI différent. Comme la redirection pourrait être altérée à cette occasion, le client DEVRAIT continuer d'utiliser l'URI de demande pour les demandes ultérieures. Cette réponse n'est mettable en antémémoire que si c'est indiqué par un en-tête Cache-Control ou Expires.

L'URI temporaire DEVRAIT être donné par le champ Location dans la réponse. Sauf si la méthode de demande était HEAD, l'entité de la réponse DEVRAIT contenir une courte note en hypertexte avec un hyperlien avec le ou les nouveaux URI.

Si le code d'état 302 est reçu en réponse à une demande autre que GET ou HEAD, l'agent d'utilisateur NE DOIT PAS automatiquement rediriger la demande sauf si elle peut être confirmée par l'utilisateur car cela pourrait changer les conditions dans lesquelles la demande a été produite.

Note Les RFC 1945 et RFC 2068 spécifient que le client n'est pas admis à changer la méthode sur la demande redirigée. Cependant, la plupart des mises en œuvre d'agent d'utilisateur traitent 302 comme si c'était une réponse 303, effectuant un GET sur la valeur de champ de Location sans considérer la méthode de la demande d'origine. Les codes d'état 303 et 307 ont été ajoutés pour des serveurs qui souhaitent rendre claire sans ambiguïté quelle sorte de réaction est attendue du client.

10.3.4 303 Voir un autre

La réponse à la demande peut être trouvée sous un URI différent et DEVRAIT être restituée en utilisant une méthode GET sur cette ressource. Cette méthode existe principalement pour permettre la sortie d'un script activé avec POST pour rediriger l'agent d'utilisateur sur une ressource choisie. Le nouvel URI n'est pas une référence de substitution à la ressource demandée à l'origine. La réponse 303 NE DOIT PAS être mise en antémémoire, mais la réponse à la seconde demande (redirigée) pourrait être mettable en antémémoire.

L'URI différent DEVRAIT être donné par le champ Location dans la réponse. Sauf si la méthode de demande était HEAD, l'entité de la réponse DEVRAIT contenir une brève note hypertexte avec un hyperlien sur le ou les nouveaux URI.

Note De nombreux agents d'utilisateur pré-HTTP/1.1 ne comprennent pas l'état 303. Lorsque l'interopérabilité avec de tels clients pose problème, le code d'état 302 peut être utilisé à la place, car la plupart des agents d'utilisateur réagissent à une réponse 302 comme décrit pour 303.

10.3.5 304 Non modifié

Si le client a effectué une demande GET conditionnelle et si l'accès est autorisé, mais que le document n'a pas été modifié, le serveur DEVRAIT répondre par ce code d'état. La réponse 304 NE DOIT PAS contenir un corps

de message, et donc elle est toujours terminée par la première ligne vide après le champ d'en-tête.

La réponse DOIT inclure les champs d'en-tête suivants :

- Date, sauf si son omission est exigée par le paragraphe 14.18.1.
- Si un serveur d'origine sans horloge obéit à ces règles, et si les mandataires et clients ajoutent leur propre Date à toute réponse reçue qui n'en a pas (comme déjà spécifié par la [RFC 2068], paragraphe 14.19), les antémémoires fonctionneront correctement.
- ETag et/ou Content-Location, si l'en-tête a été envoyé dans une réponse 200 à la même demande.
- Expires, Cache-Control, et/ou Vary, si la valeur de champ diffère de celle envoyée dans toute réponse précédente pour la même variante.

Si le GET conditionnel a utilisé un valideur fort d'antémémoire (voir au paragraphe 13.3.3), la réponse NE DEVRAIT PAS inclure d'autres en-têtes d'entité. Autrement, (c'est-à-dire, si le GET conditionnel a utilisé un valideur faible) la réponse NE DOIT PAS inclure d'autre en-tête d'entité ; ceci empêche les incohérences entre corps d'entité en antémémoire et en-têtes mis à jour.

Si une réponse 304 indique une entité qui n'est pas actuellement en antémémoire, l'antémémoire DOIT ne pas tenir compte de la réponse et répéter la demande sans la condition.

Si une antémémoire utilise une réponse 304 reçue pour mettre à jour une entrée d'antémémoire, l'antémémoire DOIT mettre à jour l'entrée pour refléter toute nouvelle valeur de champ donnée dans la réponse.

10.3.6 305 Utiliser un mandataire

L'accès à la ressource demandée DOIT être effectué à travers le mandataire donné par le champ Location. Le champ Location donne l'URI du mandataire. Le receveur est censé répéter cette seule demande via le mandataire. Les réponses 305 NE DOIVENT être générées que par les serveurs d'origine.

Note : La RFC 2068 ne disait pas clairement que 305 était destiné à rediriger une seule demande, et ne devait être généré que par les serveurs d'origine. Ne pas observer ces limitations a de sérieuses conséquences sur la sécurité.

10.3.7 306 (non utilisé)

Le code d'état 306 était utilisé dans une version précédente de la spécification, il n'est plus utilisé, et le code est réservé.

10.3.8 307 Temporairement redirigé

La ressource demandée réside temporairement sous un URI différent. Comme la redirection PEUT être altérée à cette occasion, le client DEVRAIT continuer d'utiliser l'URI de demande pour les demandes ultérieures. Cette réponse ne peut être mise en antémémoire que si c'est indiqué par un champ d'en-tête Cache-Control ou Expires.

L'URI temporaire DEVRAIT être donné par le champ Location dans la réponse. Sauf si la méthode de demande était HEAD, l'entité de la réponse DEVRAIT contenir une courte note hypertexte avec un hyperlien sur le ou les nouveaux URI, car de nombreux agents d'utilisateur pré HTTP/1.1 ne comprennent pas l'état 307. Donc, la note DEVRAIT contenir les informations nécessaires pour qu'un utilisateur répète la demande d'origine sur le nouvel URI.

Si le code d'état 307 est reçu en réponse à une demande autre que GET ou HEAD, l'agent d'utilisateur NE DOIT PAS automatiquement rediriger la demande sauf si elle peut être confirmée par l'utilisateur, car ceci pourrait changer les conditions dans lesquelles la demande a été produite.

10.4 Erreur client 4xx

La classe 4xx de code d'état est destinée aux cas dans lesquels le client semble s'être trompé. Excepté lorsqu'il répond à une demande HEAD, le serveur DEVRAIT inclure une entité contenant une explication de la situation

d'erreur, et s'il s'agit d'une condition temporaire ou permanente. Ces codes d'état sont applicables à toute méthode de demande. Les agents d'utilisateur DEVRAIENT afficher à l'utilisateur toutes les entités incluses.

Si le client envoie des données, une mise en œuvre de serveur utilisant TCP DEVRAIT être attentive à s'assurer que le client accuse réception du ou des paquets contenant la réponse, avant que le serveur ne close la connexion d'entrée. Si le client continue d'envoyer des données au serveur après la clôture, la pile TCP du serveur enverra un paquet de rétablissement au client, qui peut écraser la mémoire tampon d'entrée non acquittée du client avant qu'elles puissent être lues et interprétées par l'application HTTP.

10.4.1 400 Mauvaise demande

La demande n'a pas pu être comprise par le serveur à cause d'une syntaxe mal formée. Le client NE DEVRAIT PAS répéter la demande sans modifications.

10.4.2 401 Non autorisé

La demande exige une authentification de l'utilisateur. La réponse DOIT inclure un champ d'en-tête WWW-Authenticate (paragraphe 14.47) contenant une mise en cause applicable à la ressource demandée. Le client PEUT répéter la demande avec un champ d'en-tête Authorization convenable (paragraphe 14.8). Si la demande a déjà inclus les accréditifs d'autorisation, la réponse 401 indique alors que l'autorisation a été refusée pour ces accréditifs. Si la réponse 401 contenait la même mise en cause que la réponse précédente, et si l'agent d'utilisateur a déjà essayé l'authentification au moins une fois, l'utilisateur DEVRAIT être alors mis en présence de l'entité qui est donnée dans la réponse, car cette entité peut inclure des informations de diagnostic pertinentes. L'authentification d'accès HTTP est expliquée dans "Authentification HTTP : Authentification d'accès de base et par résumé" [43].

10.4.3 402 Paiement exigé

Ce code est réservé pour une utilisation future.

10.4.4 403 Interdit

Le serveur a compris la demande, mais refuse de la satisfaire. L'autorisation ne sert à rien et la demande NE DEVRAIT PAS être répétée. Si la méthode de la demande n'était pas HEAD et si le serveur souhaite rendre publique la raison pour laquelle la demande n'a pas été satisfaite, il DEVRAIT décrire la raison du refus dans l'entité. Si le serveur ne souhaite pas rendre ces informations disponibles au client, le code d'état 404 (Pas trouvé) peut être utilisé à la place.

10.4.5 404 Pas trouvé

Le serveur n'a rien trouvé qui corresponde à l'URI de demande. Aucune indication n'est donnée sur le fait que la condition soit temporaire ou permanente. Le code d'état 410 (Parti) DEVRAIT être utilisé si le serveur sait, par un mécanisme configurable en interne, qu'une ancienne ressource est indisponible de façon permanente et n'a pas d'adresse de renvoi. Ce code d'état est couramment utilisé lorsque le serveur ne souhaite pas révéler exactement pourquoi la demande a été refusée, ou lorsque aucune autre réponse n'est applicable.

10.4.6 405 Méthode non admise

La méthode spécifiée dans la ligne de demande n'est pas admise pour la ressource identifiée par l'URI de demande. La réponse DOIT inclure un en-tête Allow contenant une liste des méthodes valides pour la ressource demandée.

10.4.7 406 Non acceptable

La ressource identifiée par la demande est seulement capable de générer des entités de réponse qui ont des caractéristiques de contenu non acceptables selon les en-têtes d'acceptation envoyés dans la demande.

Sauf s'il s'agit d'une demande HEAD, la réponse DEVRAIT inclure une entité contenant une liste des caractéristiques et localisations d'entité disponibles entre lesquelles l'utilisateur ou agent d'utilisateur peut choisir celle qui est la plus appropriée. Le format d'entité est spécifié par le type de support donné dans le champ d'en-tête Content-Type. Selon le format et les capacités de l'agent d'utilisateur, le choix le plus approprié PEUT être effectué automatiquement. Cependant, la présente spécification ne définit aucune norme pour un tel choix automatique.

Note : Les serveurs HTTP/1.1 sont autorisés à retourner des réponses qui ne sont pas acceptables selon les en-têtes d'acceptation envoyés dans la demande. Dans certains cas, ceci peut même être préférable à l'envoi d'une réponse 406. Les agents d'utilisateur sont invités à inspecter les en-têtes d'une réponse entrante pour déterminer si elle est acceptable.

Si la réponse pourrait être inacceptable, un agent d'utilisateur DEVRAIT temporairement arrêter la réception des données et interroger l'utilisateur sur la décision à prendre quant aux actions ultérieures.

10.4.8 407 Authentification du mandataire exigée

Ce code est similaire à 401 (Non autorisé), mais indique que le client doit d'abord s'authentifier auprès du mandataire. Le mandataire DOIT retourner un champ d'en-tête Proxy-Authenticate (paragraphe 14.33) contenant une mise en cause applicable au mandataire pour la ressource demandée. Le client PEUT répéter la demande avec un champ d'en-tête Proxy-Authorization convenable (paragraphe 14.34). L'authentification d'accès HTTP est expliquée dans "Authentification HTTP : Authentification d'accès de base et par résumé" [43].

10.4.9 408 Expiration du délai de la demande

Le client n'a pas produit une demande dans le délai pendant lequel serveur était prêt à attendre. Le client PEUT répéter la demande sans modification à tout moment ultérieur.

10.4.10 409 Conflit

La demande n'a pas pu être achevée à cause d'un conflit avec l'état en cours de la ressource. Ce code n'est admis que dans des situations où on s'attend à ce que l'utilisateur soit capable de résoudre le conflit et soumettre à nouveau la demande. Le corps de la réponse DEVRAIT inclure suffisamment d'informations pour que l'utilisateur reconnaisse la source du conflit. En principe, l'entité de réponse devrait inclure des informations suffisantes pour que l'utilisateur ou l'agent d'utilisateur résolve le problème ; cependant, cela peut n'être pas possible et n'est pas exigé.

Les conflits vont le plus vraisemblablement survenir en réponse à une demande PUT. Par exemple, si le numéro de version devait être utilisé et si l'entité dans PUT inclut des changements à une ressource qui sont en conflit avec ceux faits par une demande précédente (de tierce partie), le serveur pourrait utiliser la réponse 409 pour indiquer qu'il ne peut achever la demande. Dans ce cas, l'entité de réponse va vraisemblablement contenir une liste des différences entre les deux versions dans un format défini par le type de contenu de la réponse.

10.4.11 410 Parti

La ressource demandée n'est plus disponible au serveur et aucune adresse de retransmission n'est connue. Cette condition est en principe considérée comme permanente. Les clients avec des capacités d'édition de liens DEVRAIENT supprimer les références à l'URI de demande après approbation de l'utilisateur. Si le serveur ne sait pas, ou n'a pas de possibilité de déterminer si la condition est permanente ou non, le code d'état 404 (Non trouvé) DEVRAIT être utilisé à la place. Cette réponse peut être mise en antémémoire sauf indication contraire.

La réponse 404 est principalement destinée à aider aux tâches de maintenance de la toile en notifiant au receveur que la ressource est intentionnellement indisponible et que les propriétaires du serveur désirent que les liens distants à cette ressource soient supprimés. Un tel événement est courant pour une durée limitée, des services promotionnels et pour des ressources appartenant à des individus ne travaillant plus sur le site du serveur. Il n'est pas nécessaire de marquer toutes les ressources indisponibles de façon permanente comme "parties" ou de conserver la marque pour une durée déterminée -- ceci est laissé à la discrétion du propriétaire

du serveur.

10.4.12 411 Longueur requise

Le serveur refuse d'accepter la demande sans une longueur de contenu définie. Le client PEUT répéter la demande si il ajoute un champ d'en-tête Content-Length valide contenant la longueur du corps de message dans le message de demande.

10.4.13 412 Échec de précondition

La précondition donnée dans un ou plusieurs des champs d'en-tête de la demande a été évaluée à faux lorsqu'ils ont été testés sur le serveur. Ce code de réponse permet au client de fixer des préconditions sur les méta-informations (données de champ d'en-tête) de la ressource en cours et empêcher ainsi que la méthode demandée ne soit appliquée à une ressource autre que celle désirée.

10.4.14 413 Entité de demande trop grande

Le serveur refuse de traiter une demande parce que l'entité de demande est plus grande que ce que le serveur est désireux ou capable de traiter. Le serveur PEUT clore la connexion pour empêcher le client de poursuivre la demande.

Si la condition est temporaire, le serveur DEVRAIT inclure un champ d'en-tête Retry-After *(réessayer plus tard)* pour indiquer que c'est temporaire et après quel délai le client PEUT essayer à nouveau.

10.4.15 414 URI de demande trop long

Le serveur refuse de servir la demande parce que l'URI de demande est plus long que ce que le serveur veut interpréter. Cette condition rare ne surviendra vraisemblablement que lorsqu'un client a improprement converti une demande POST en demande GET avec de longues informations d'interrogation, lorsque le client est descendu dans un URI de redirection "trou noir" (par exemple, un préfixe d'URI redirigé qui pointe sur un suffixe de lui-même), ou lorsque le serveur est soumis à une attaque par un client qui essaye d'exploiter des trous dans la sécurité présents sur certains serveurs utilisant des mémoires tampon de longueur fixe pour lire ou manipuler l'URI de demande.

10.4.16 415 Type de support non pris en charge

Le serveur refuse de servir la demande parce que l'entité de la demande est dans un format non pris en charge par la ressource demandée pour la méthode demandée.

10.4.17 416 Gamme demandée non acceptable

Un serveur DEVRAIT retourner une réponse avec ce code d'état si une demande inclut un champ d'en-tête de demande Range (paragraphe 14.35) et qu'aucune des valeurs de spécification de gamme de ce champ ne recouvre la dimension actuelle de la ressource choisie, et que la demande n'inclut pas un champ d'en-tête de demande If-Range. (Pour les gammes d'octet, cela signifie que les valeurs first-byte-pos de toutes les valeurs byte-range-spec sont supérieures à la longueur actuelle de la ressource choisie.)

Lorsque ce code d'état est retourné pour une demande byte-range, la réponse DEVRAIT inclure un champ d'en-tête d'entité Content-Range qui spécifie la longueur actuelle de la ressource choisie (voir au paragraphe 14.16). Cette réponse NE DOIT PAS utiliser le type de contenu multipart/byteranges.

10.4.18 417 Échec de l'attente

L'attente donnée dans un champ d'en-tête de demande Expect (voir au paragraphe 14.20) n'a pas pu être satisfaite par ce serveur, ou, si le serveur est un mandataire, le serveur a vu de façon non ambiguë que la demande ne pourra pas être satisfaite par le serveur du prochain bond.

10.5 Erreur du serveur 5xx

Les codes d'état de réponse commençant par le chiffre "5" indiquent les cas dans lesquels le serveur sait qu'il a fait une erreur ou est incapable de traiter la demande. Excepté lors de la réponse à une demande HEAD, le serveur DEVRAIT inclure une entité contenant une explication de la situation d'erreur, et si c'est une condition temporaire ou permanente. Les agents d'utilisateur DEVRAIENT afficher toute entité incluse à l'utilisateur. Ces codes de réponse sont applicables à toute méthode de demande.

10.5.1 500 Erreur interne du serveur

Le serveur a rencontrée une condition inattendue qui l'empêche de satisfaire la demande.

10.5.2 501 Non mise en œuvre

Le serveur ne prend pas en charge la fonctionnalité exigée pour satisfaire la demande. C'est la réponse appropriée lorsque le serveur ne reconnaît pas la méthode de la demande et n'est capable de la prendre en charge pour aucune ressource.

10.5.3 502 Mauvaise passerelle

Le serveur, tout en agissant comme passerelle ou mandataire, a reçu une réponse invalide de la part du serveur amont auquel il a accédé en essayant de satisfaire la demande.

10.5.4 503 Service indisponible

Le serveur est incapable de traiter actuellement la demande à cause d'une surcharge temporaire ou de la maintenance du serveur. Cela implique qu'il s'agit d'une condition temporaire qui sera levée après un certain temps. Si elle est connue, la longueur du délai PEUT être indiquée dans un en-tête Retry-After. Si aucun Retry-After n'est donné, le client DEVRAIT traiter la réponse comme il le ferait pour une réponse 500.

Note L'existence du code d'état 503 n'implique pas qu'un serveur doive l'utiliser lorsqu'il est encombré. Certains serveurs peuvent simplement souhaiter refuser la connexion.

10.5.5 504 Temporisation de passerelle expirée

Le serveur, agissant comme passerelle ou mandataire, n'a pas reçu à temps une réponse du serveur amont spécifié par l'URI (par exemple HTTP, FTP, LDAP) ou autre serveur auxiliaire (par exemple DNS) auquel il a besoin d'accéder pour essayer de mener la demande à bien.

Note pour les développeurs : certains mandataires actuels sont réputés retourner 400 ou 500 lorsque le DNS arrive en fin de temporisation.

10.5.6 505 Version HTTP non prise en charge

Le serveur ne prend pas en charge, ou refuse de prendre en charge, la version de protocole HTTP qui a été utilisée dans le message de demande. Le serveur indique qu'il n'est pas capable ou n'est pas désireux de mener à bien la demande en utilisant la même version majeure que le client, comme décrit au paragraphe 3.1, au moyen de ce message d'erreur. La réponse DEVRAIT contenir une entité décrivant les raisons de la non prise en charge de cette version et quels autres protocoles sont pris en charge par ce serveur.

11 Authentification d'accès

HTTP fournit plusieurs mécanismes d'authentification de question-réponse FACULTATIFS qui peuvent être

utilisés par un serveur pour mettre en cause la demande d'un client et par un client pour fournir les informations d'authentification. Le cadre général de l'authentification d'accès, et la spécification de l'authentification "de base" et "par résumé", sont spécifiés dans "Authentification HTTP : Authentification d'accès de base et par résumé" [43]. La présente spécification adopte les définitions de "mise en cause" et "accréditifs" de cette spécification.

12 Négociation du contenu

La plupart des réponses HTTP comportent une entité qui contient des informations destinées à l'utilisateur humain. Naturellement, il est souhaitable de fournir à l'utilisateur la "meilleure entité disponible" correspondant à la demande. Malheureusement pour les serveurs et les antémémoires, tous les utilisateurs n'ont pas les mêmes préférences au sujet de ce qui est "le meilleur" et tous les agents d'utilisateur sont également capables de rendre tous les types d'entité. Pour cette raison, HTTP a des dispositions pour plusieurs des mécanismes de "négociation de contenu" -- le processus de choix de la meilleure représentation pour une réponse donnée lorsque plusieurs représentations sont disponibles.

Note Ceci ne s'appelle pas "négociation de format" parce que différentes représentations peuvent être du même type de support, mais utiliser de capacités différentes de ce type, être dans des langues différentes, etc.

Toute réponse contenant un corps d'entité PEUT être soumise à négociation, y compris les réponses d'erreur.

Deux sortes de négociation de contenu sont possibles dans HTTP : la négociation conduite par le serveur et la négociation conduite par l'agent. Ces deux sortes de négociation sont orthogonales et peuvent donc être utilisées séparément ou en combinaison. Une méthode de combinaison, appelée négociation transparente, survient lorsque une antémémoire utilise les informations de négociation conduites par l'agent fournies par le serveur d'origine afin d'offrir une négociation conduite par le serveur pour les demandes ultérieures.

12.1 Négociation conduite par le serveur

Si le choix de la meilleure représentation pour une réponse est fait par un algorithme situé au serveur, on l'appelle une négociation conduite par le serveur. Le choix se fonde sur les représentations disponibles de la réponse (les dimensions selon lesquelles elle peut varier ; par exemple, le langage, le codage du contenu, etc.) et les contenus des champs d'en-tête particuliers dans le message de demande ou sur d'autres informations relatives à la demande (telles que l'adresse réseau du client).

La négociation conduite par le serveur est avantageuse lorsque l'algorithme de choix parmi les représentations disponibles est difficile à décrire à l'agent d'utilisateur, ou lorsque le serveur désire envoyer ses "meilleures offres" au client avec la première réponse (souhaitant éviter le délai d'aller-retour d'une demande ultérieure si la "meilleure offre" est acceptable par l'utilisateur). Pour améliorer les offres du serveur, l'agent d'utilisateur PEUT inclure des champs d'en-tête de demande (Accept, Accept-Language, Accept-Encoding, etc.) qui décrivent ses préférences pour une telle réponse.

La négociation conduite par le serveur a des inconvénients :

1. Il est impossible que le serveur détermine avec précision ce qui peut être "meilleur" pour un utilisateur donné, car cela exigerait une connaissance complète à la fois des capacités de l'agent d'utilisateur et de l'utilisation prévue de la réponse (par exemple, l'utilisateur veut-il la voir affichée à l'écran ou imprimée sur papier ?).
2. Que l'agent d'utilisateur décrive ses capacités dans chaque demande peut être à la fois très inefficace (étant donné que seul un faible pourcentage des réponses a des représentations multiples) et une violation potentielle de la confidentialité de l'utilisateur.
3. Elle complique la mise en œuvre d'un serveur d'origine et les algorithmes de génération des réponses à une demande.
4. Elle peut limiter la capacité d'une antémémoire publique à utiliser la même réponse pour plusieurs demandes de l'utilisateur.

HTTP/1.1 inclut les champs d'en-tête de demande suivants pour activer la négociation conduite par le serveur à travers la description des capacités de l'agent d'utilisateur et des préférences de l'utilisateur : Accept (paragraphe 14.1), Accept-Charset (paragraphe 14.2), Accept-Encoding (paragraphe 14.3), Accept-Language (paragraphe 14.4), et User-Agent (paragraphe 14.43). Cependant, un serveur d'origine n'est pas limité à ces

dimensions et PEUT varier les réponses sur la base de tout aspect de la demande, y compris des informations en-dehors des champs d'en-tête de la demande ou au sein de champs d'en-tête d'extension non définis par la présente spécification.

Le champ d'en-tête Vary peut être utilisé pour exprimer les paramètres que le serveur utilise pour choisir une représentation qui est soumise à une négociation conduite par le serveur. Voir au paragraphe 13.6 l'utilisation du champ d'en-tête Vary par les antémémoires et au paragraphe 14.44 l'utilisation du champ d'en-tête Vary par les serveurs.

12.2 Négociation conduite par l'agent

Avec la négociation conduite par l'agent, le choix de la meilleure représentation d'une réponse est effectué par l'agent d'utilisateur après avoir reçu une réponse initiale de la part du serveur d'origine. La sélection est fondée sur une liste des représentations disponibles de la réponse incluses dans les champs d'en-tête ou corps d'entité de la réponse initiale, chaque représentation étant identifiée par son propre URI. Le choix parmi les représentations peut être effectué automatiquement (si l'agent d'utilisateur est capable de le faire) ou manuellement par l'usager qui choisit dans un menu généré (éventuellement par hypertexte).

La négociation conduite par l'agent est avantageuse lorsque la réponse peut varier au delà des dimensions communément utilisées (comme le type, la langue, ou le codage) lorsque le serveur d'origine est incapable de déterminer les capacités d'un agent d'utilisateur à examiner la demande, et généralement lorsque des antémémoires publiques sont utilisées pour répartir la charge du serveur et réduire l'utilisation du réseau.

La négociation conduite par l'agent souffre de l'inconvénient d'avoir besoin d'une seconde demande pour obtenir la meilleure représentation. Cette seconde demande n'est efficace que lorsque la mise en antémémoire est utilisée. De plus, la présente spécification ne définit aucun mécanisme pour prendre en charge la sélection automatique, quoiqu'elle n'empêche aucun de ces mécanismes d'être développé comme extension et utilisé dans HTTP/1.1.

HTTP/1.1 définit les codes d'état 300 (Choix multiple) et 406 (Non acceptable) pour permettre la négociation conduite par l'agent lorsque le serveur ne veut pas ou n'est pas capable de fournir une variante de réponse en utilisant la négociation conduite par le serveur.

12.3 Négociation transparente

La négociation transparente est une combinaison des deux négociations, conduite par le serveur et conduite par l'agent. Lorsque une antémémoire est alimentée avec une forme de la liste des représentations disponibles de la réponse (comme dans la négociation conduite par l'agent) et que les dimensions de variance sont entièrement comprises par l'antémémoire, celle-ci devient capable d'effectuer la négociation conduite par le serveur au nom du serveur d'origine pour les demandes ultérieures sur cette ressource.

La négociation transparente présente l'avantage de répartir le travail de négociation qui autrement serait exigé du serveur d'origine et aussi de supprimer le délai de seconde demande de la négociation conduite par l'agent lorsque l'antémémoire est capable de deviner correctement la bonne réponse.

La présente spécification ne définit aucun mécanisme pour la négociation transparente, quoiqu'elle n'empêche pas qu'un tel mécanisme soit développé comme extension pouvant être utilisée dans HTTP/1.1.

13 Mise en antémémoire dans HTTP

HTTP est normalement utilisé pour des systèmes d'information répartis, où les performances peuvent être améliorées par l'utilisation d'antémémoires de réponse. Le protocole HTTP/1.1 comporte un certain nombre d'éléments destinés à effectuer le travail de mise en antémémoire aussi bien que possible. Parce que ces éléments sont inextricables d'autres aspects du protocole, et parce qu'ils interagissent les uns avec les autres, il est utile de décrire le concept de base de la mise en antémémoire de HTTP séparément de la description détaillée des méthodes, en-têtes, codes de réponse, etc.

La mise en antémémoire serait inutile si elle n'améliorait pas significativement les performances. L'objectif de la

mise en antémémoire dans HTTP/1.1 est d'éliminer le besoin d'envoyer des demandes dans de nombreux cas, et d'éliminer le besoin d'envoyer les réponses complètes dans de nombreux autres cas. Le premier réduit le nombre d'allers-retours du réseau exigé pour nombre d'opérations ; on utilise un mécanisme d'"expiration" à cette fin (voir au paragraphe 13.2). Le dernier réduit les exigences de bande passante du réseau ; on utilise un mécanisme de "validation" à cette fin (voir au paragraphe 13.3).

Les exigences de performances, disponibilité, et d'opérations déconnectées nous obligent à être capables d'assouplir l'objectif de transparence sémantique. Le protocole HTTP/1.1 permet aux serveurs d'origine, antémémoires et clients, de réduire explicitement la transparence quand nécessaire. Cependant, à cause d'opérations non transparentes qui peuvent amener de la confusion chez les utilisateurs non experts, et qu'il peut être incompatible avec certaines applications de serveur (comme celles pour les commandes de marchandises) le protocole exige que la transparence soit assouplie

- seulement par une demande explicite de niveau protocole lorsqu'elle est assouplie par le client ou serveur d'origine
- seulement par un avertissement explicite à l'utilisateur terminal lorsqu'elle est assouplie par l'antémémoire ou le client.

Donc, le protocole HTTP/1.1 fournit ces éléments importants :

1. Les caractéristiques du protocole qui fournissent la transparence sémantique complète quand c'est exigé par toutes les parties.
2. Les caractéristiques du protocole qui permettent à un serveur d'origine ou agent d'utilisateur de demander explicitement et de contrôler une opération non transparente.
3. Les caractéristiques du protocole qui permettent à une antémémoire d'adjoindre des avertissements aux réponses qui ne préservent pas l'approximation demandée de transparence sémantique.

Un principe de base est qu'il doit être possible pour les clients de détecter tout assouplissement potentiel de la transparence sémantique.

Note : Le développeur de serveur, antémémoire, ou client peut être amené à faire face à des décisions de conception qui ne sont pas exposées explicitement dans la présente spécification. Si une décision risque d'affecter la transparence sémantique, le développeur ne doit pas hésiter sur la question du maintien de la transparence si une analyse attentive et complète montre des avantages significatifs à rompre la transparence.

13.1.1 Ce qui est correct pour une antémémoire

Une antémémoire correcte DOIT répondre à une demande avec la réponse la plus à jour détenue par l'antémémoire qui est appropriée à la demande (voir les paragraphes 13.2.5, 13.2.6, et 13.12) qui satisfait une des conditions suivantes :

1. Elle a été vérifiée quant à son équivalence avec ce que le serveur d'origine aurait retourné en revalidant la réponse avec le serveur d'origine (paragraphe 13.3) ;

2. Elle est "assez fraîche" (voir au paragraphe 13.2). Dans le cas par défaut, cela signifie qu'elle satisfait au moins aux exigences restrictives sur la fraîcheur du client, du serveur d'origine, et de l'antémémoire (voir au paragraphe 14.9) ; si le serveur d'origine le spécifie, c'est la seule exigence de fraîcheur du serveur d'origine.

   Si une réponse mémorisée n'est pas "assez fraîche" selon l'exigence de fraîcheur la plus restrictive à la fois du client et du serveur d'origine, dans des circonstances bien étudiées, l'antémémoire PEUT encore retourner la réponse avec l'en-tête Warning approprié (voir les paragraphes 13.1.5 et 14.46), sauf si une telle réponse est interdite (par exemple, par une directive d'antémémoire "pas de mémorisation", ou par une directive de demande d'antémémoire "pas d'antémémoire" ; voir au paragraphe 14.9).

3. C'est un message de réponse 304 (Non modifié), 305 (Rediriger le mandataire), ou d'erreur (4xx ou 5xx) approprié.

Si l'antémémoire ne peut pas communiquer avec le serveur d'origine, une antémémoire correcte DEVRAIT répondre comme ci-dessus si la réponse peut être correctement servie depuis l'antémémoire ; sinon elle DOIT retourner une erreur ou un avertissement indiquant qu'il y a eu un échec de communication.

Si une antémémoire reçoit une réponse (une réponse entière, ou une réponse 304 (Non modifiée)) qu'elle devrait normalement transmettre au client demandeur, et si la réponse reçue n'est plus fraîche, l'antémémoire DEVRAIT la transmettre au client demandeur sans ajouter de nouveau Warning (mais sans retirer l'en-tête Warning existant). Une antémémoire NE DEVRAIT PAS essayer de revalider une réponse simplement parce que la réponse est devenue périmée dans le transit ; ceci pourrait conduire à une boucle sans fin. Un agent d'utilisateur qui reçoit une réponse périmée sans un Warning PEUT afficher une indication d'avertissement à l'utilisateur.

13.1.2 Avertissements

Chaque fois qu'une antémémoire retourne une réponse qui n'est ni de première main ni "assez fraîche" (au sens de la condition 2 du paragraphe 13.1.1) elle DOIT adjoindre un avertissement à cet effet, en utilisant un en-tête général Warning. L'en-tête Warning et les avertissements actuellement définis sont décrits au paragraphe 14.46. L'avertissement permet aux clients d'entreprendre l'action appropriée.

Les avertissements PEUVENT être utilisés à d'autres fins, à la fois en rapport avec l'antémémoire et pour d'autres raisons. L'utilisation d'un avertissement, plutôt que d'un code d'état d'erreur, distingue ces réponses des véritables défaillances.

Des codes d'avertissement à trois chiffres sont alloués aux avertissements. Le premier chiffre indique si l'avertissement DOIT ou NE DOIT PAS être supprimé d'une entrée d'antémémoire mémorisée après une revalidation réussie :

- 1xx Avertissements qui décrivent l'état de fraîcheur ou de revalidation de la réponse, et donc DOIVENT être supprimés après une revalidation réussie. Les codes d'avertissement 1XX ne PEUVENT être générés par une antémémoire que lors de la validation d'une entrée d'antémémoire. Ils NE DOIVENT PAS être générés par les clients.

- 2xx Avertissements qui décrivent certains aspects du corps d'entité ou d'en-tête d'entité qui ne sont pas rectifiés par une revalidation (par exemple, une compression avec perte des corps d'entité) et qui NE DOIVENT PAS être supprimés après une revalidation réussie.

Voir au paragraphe 14.46 les définitions des codes eux-mêmes.

Les antémémoires de HTTP/1.0 vont mettre en antémémoire tous les avertissements dans les réponses, sans supprimer ceux de la première catégorie. Les avertissements dans les réponses qui sont passés par des antémémoires de HTTP/1.0 portent un champ supplémentaire de date d'avertissement qui empêche un futur receveur HTTP/1.1 de croire à un avertissement mis par erreur en antémémoire.

Les avertissements portent aussi un texte d'avertissement. Le texte PEUT être en tout langage naturel approprié (peut-être sur la base des en-têtes Accept du client), et inclure une indication FACULTATIVE du jeu de caractères qui est utilisé.

Plusieurs avertissements PEUVENT être attachés à une réponse (soit par le serveur d'origine soit par une antémémoire) y compris plusieurs avertissements avec le même numéro de code. Par exemple, un serveur pourrait fournir le même avertissement en anglais et en basque.

Lorsque plusieurs avertissements sont attachés à une réponse, il peut n'être pas pratique ou raisonnable de les afficher tous à l'utilisateur. La présente version de HTTP ne spécifie pas de règles de priorité strictes pour décider quels avertissements afficher et dans quel ordre, mais suggère une certaine heuristique.

13.1.3 Mécanismes de contrôle d'antémémoire

Les mécanismes de base d'antémémoire dans HTTP/1.1 (durées d'expiration spécifiées par le serveur et valideurs) sont des directives implicites pour les antémémoires. Dans certains cas, un serveur ou client peut avoir besoin de fournir des directives explicites aux antémémoires HTTP. On utilise à cette fin l'en-tête Cache-Control.

L'en-tête Cache-Control permet à un client ou serveur de transmettre diverses directives dans des demandes

ou des réponses. Ces directives outrepassent normalement les algorithmes de mise en antémémoire par défaut. En règle générale, si il y a un conflit apparent entre les valeurs d'en-tête, c'est l'interprétation la plus restrictive qui s'applique (c'est à dire, celle qui va le plus vraisemblablement préserver la transparence sémantique). Cependant, dans certains cas, les directives de commande d'antémémoire sont explicitement spécifiées comme affaiblissant l'approximation de la transparence sémantique (par exemple, "max-stale" ou "public").

Les directives de commande d'antémémoire sont décrites en détail au paragraphe 14.9.

13.1.4 Avertissements explicites d'agent d'utilisateur

De nombreux agents d'utilisateurs rendent possible à l'utilisateur d'outrepasser les mécanismes de base de mise en antémémoire. Par exemple, l'agent d'utilisateur peut permettre à l'utilisateur de spécifier que les entités mises en antémémoire (même celles explicitement périmées) ne sont jamais validées. Ou l'agent d'utilisateur peut habituellement ajouter "Cache-Control: max-stale=3600" à chaque demande. L'agent d'utilisateur NE DEVRAIT PAS revenir par défaut au comportement non transparent, ou au comportement qui a pour résultat une mise en antémémoire anormalement inefficace, mais PEUT être explicitement configuré pour le faire par une action explicite de l'utilisateur.

Si l'utilisateur a outrepassé les mécanismes de base de mise en antémémoire, l'agent d'utilisateur DEVRAIT explicitement indiquer à l'utilisateur lorsque cela résulte en l'affichage d'informations qui pourraient ne pas satisfaire aux exigences de transparence du serveur (en particulier, si l'entité affichée est connue pour être périmée). Comme le protocole permet normalement à l'agent d'utilisateur de déterminer si les réponses sont ou non périmées, cette indication n'a besoin d'être affichée que lorsque ceci survient réellement. L'indication ne doit pas nécessairement être une boîte de dialogue ; elle pourrait être une icône (par exemple, le dessin d'un poisson pourri) ou quelque autre indicateur.

Si l'utilisateur a outrepassé les mécanismes de mise en antémémoire d'une façon qui pourrait réduire anormalement l'efficacité des antémémoires, l'agent d'utilisateur DEVRAIT indiquer de façon continue cet état à l'utilisateur (par exemple, par l'affichage d'un dessin de billets de banque en flammes) de sorte que l'utilisateur ne consomme pas par inadvertance des ressources excessives ou ne souffre d'un délai de latence excessif.

13.1.5 Exceptions aux règles et avertissements

Dans certains cas, l'opérateur d'une antémémoire PEUT choisir de la configurer de façon à retourner les réponses périmées même lorsque ce n'est pas demandé par les clients. Cette décision ne devrait pas être prise à la légère, mais peut être nécessaire pour des raisons de disponibilité ou de performances, particulièrement lorsque l'antémémoire est mal connectée au serveur d'origine. Chaque fois qu'une antémémoire retourne une réponse périmée, elle DOIT la marquer comme telle (en utilisant un en-tête Warning) pour permettre au logiciel client d'alerter l'utilisateur qu'il pourrait y avoir un problème potentiel.

Il permet aussi à l'agent d'utilisateur de prendre des mesures pour obtenir des réponses fraîches ou de première main. Pour cette raison, une antémémoire NE DEVRAIT PAS retourner de réponse périmée si le client demande explicitement des réponses fraîches ou de première main, sauf si c'est impossible de le faire pour des raisons techniques ou de politique.

13.1.6 Comportement commandé par le client

Alors que le serveur d'origine (et dans une moindre mesure, les antémémoires intermédiaires, par leur contribution au vieillissement d'une réponse) sont les sources principales des informations d'expiration, dans certains cas, le client peut avoir besoin de contrôler la décision d'une antémémoire sur le point de savoir s'il faut retourner une réponse mise en mémoire sans la valider. Les clients font cela en utilisant plusieurs directives de l'en-tête Cache-Control.

La demande d'un client PEUT spécifier l'âge maximum qu'il accepte pour une réponse non validée ; spécifier qu'une valeur de zéro force le ou les antémémoires à revalider toutes les réponses. Un client PEUT aussi spécifier la durée minimum restante avant qu'une réponse arrive à expiration. Ces deux options accroissent les contraintes sur le comportement des antémémoires, et ne permettent pas d'assouplir davantage l'approximation de la transparence sémantique par l'antémémoire.

Un client PEUT aussi spécifier qu'il va accepter les réponses périmées, jusqu'à un certain degré maximum de péremption. Ceci desserre les contraintes pesant sur les antémémoires, et peut ainsi violer les contraintes spécifiées par le serveur d'origine sur la transparence sémantique, mais peut être nécessaire pour prendre en charge le fonctionnement déconnecté, ou la forte disponibilité face à une connexité déficiente.

13.2 Modèle d'expiration

13.2.1 Expiration spécifiée par le serveur

La mise en antémémoire de HTTP fonctionne mieux lorsque les antémémoires peuvent éviter entièrement de faire des demandes au serveur d'origine. Le principal mécanisme pour éviter des demandes est pour un serveur d'origine de fournir un temps d'expiration explicite dans le futur, indiquant qu'une réponse PEUT être utilisée pour satisfaire des demandes ultérieures. En d'autres termes, une antémémoire peut retourner une réponse fraîche sans contacter d'abord le serveur.

On s'attend à ce que les serveurs allouent de futurs temps d'expiration explicites aux réponses dans l'espoir que l'entité ne va vraisemblablement pas changer, d'une façon sémantiquement significative, avant d'atteindre le délai d'expiration. Ceci préserve normalement la transparence sémantique, pour autant que les délais d'expiration du serveur soient choisis avec soin.

Le mécanisme d'expiration ne s'applique qu'aux réponses tirées d'une antémémoire et non aux réponses de première main transmises immédiatement au client demandeur.

Si un serveur d'origine souhaite forcer une antémémoire sémantiquement transparente à valider chaque demande, il PEUT allouer un temps d'expiration explicite dans le passé. Cela signifie que la réponse est toujours périmée, et ainsi l'antémémoire DEVRAIT la valider avant de l'utiliser pour des demandes ultérieures. Voir au paragraphe 14.9.4 une façon plus restrictive de forcer la revalidation.

Si un serveur d'origine souhaite forcer une antémémoire HTTP/1.1, quelle que soit sa configuration, à valider chaque demande, il DEVRAIT utiliser la directive de commande d'antémémoire "must-revalidate" (paragraphe 14.9).

Les serveurs spécifient des temps d'expiration explicites en utilisant soit l'en-tête Expires, soit la directive max-âge de l'en-tête Cache-Control.

Un délai d'expiration ne peut être utilisé pour forcer un agent d'utilisateur à rafraîchir son affichage ou recharger une ressource ; sa sémantique ne s'applique qu'aux mécanismes de mise en antémémoire, et de tels mécanismes n'ont besoin de vérifier l'état d'expiration d'une ressource que quand est initiée une nouvelle demande pour cette ressource. Voir au paragraphe 13.13 une explication de la différence entre les mécanismes d'antémémoire et d'historique.

13.2.2 Expiration heuristique

Comme les serveurs d'origine ne fournissent pas toujours des temps d'expiration explicites, les antémémoires HTTP fournissent en fait des temps d'expiration heuristiques, utilisant des algorithmes qui emploient d'autres valeurs d'en-tête (telles que la dernière heure modifiée) pour estimer une heure d'expiration plausible. La spécification HTTP/1.1 ne fournit pas d'algorithmes spécifiques, mais impose des contraintes de plus mauvais cas sur leurs résultats. Comme les temps d'expiration heuristiques pourraient compromettre la transparence sémantique, ils devraient être utilisés avec précaution, et on invite les serveurs d'origine à fournir des temps d'expiration explicites autant que possible.

13.2.3 Calculs de vieillissement

Afin de savoir si une entrée d'antémémoire est fraîche, une antémémoire a besoin de savoir si son âge excède la durée de vie de sa fraîcheur. On expose comment calculer cette dernière au paragraphe 13.2.4 ; on décrit ici comment calculer l'âge d'une réponse ou d'une entrée d'antémémoire.

Dans cet exposé, on utilise le terme "maintenant" pour signifier "la valeur actuelle de l'horloge de l'hôte qui

effectue le calcul." Les hôtes qui utilisent HTTP, mais particulièrement les hôtes qui hébergent les serveurs d'origine et les antémémoires, DEVRAIENT utiliser NTP [28] ou un protocole similaire pour synchroniser leur horloges avec une heure standard mondiale précise.

HTTP/1.1 exige que les serveurs d'origine envoient un en-tête Date, si possible, avec chaque réponse, donnant l'heure à laquelle la réponse a été générée (voir au paragraphe 14.18). On utilise le terme "date_valeur" pour noter la valeur de l'en-tête Date, sous une forme appropriée aux opérations arithmétiques.

HTTP/1.1 utilise l'en-tête de réponse Age pour convoyer l'âge estimé du message de réponse lorsqu'il est obtenu d'une antémémoire. La valeur du champ Age est l'estimation de l'antémémoire de la quantité de temps écoulé depuis que la réponse a été générée ou revalidée par le serveur d'origine.

Par nature, la valeur de Age est la somme des temps pendant lesquels la réponse a résidé dans chacune des antémémoires le long du chemin venant du serveur d'origine, plus la durée pendant laquelle elle a été en transit le long des chemins du réseau.

On utilise le terme "âge_valeur" pour noter la valeur de l'en-tête Age, sous une forme appropriée aux opérations arithmétiques.

Un âge de réponse peut être calculé de deux façons entièrement indépendantes :

1. temps présent moins la date_valeur, si l'horloge locale est raisonnablement bien synchronisée à l'horloge du serveur d'origine. Si le résultat est négatif, le résultat est remplacé par zéro.
2. âge_valeur, si toutes les antémémoires le long du chemin de la réponse mettent en œuvre HTTP/1.1.

Étant donné que nous avons deux façons indépendantes de calculer l'âge d'une réponse quand elle est reçue, on peut les combiner de la façon suivante :

corrected_received_age = max(maintenant - date_valeur, âge_valeur)

et tant que nous avons des horloges presque synchronisées ou des chemins tout HTTP/1.1, on obtient un résultat fiable (prudent).

À cause des délais imposés par le réseau, un intervalle significatif peut s'écouler entre le moment où un serveur génère une réponse et le moment où elle est reçue à la prochaine antémémoire extérieure ou client. Si il n'est pas corrigé, ce délai peut conduire à des âges trop faibles.

Comme la demande qui a donné la valeur d'âge retournée doit avoir été initiée avant la génération de la valeur Age, on peut corriger des délais imposés par le réseau en enregistrant l'heure à laquelle la demande a été initiée. Puis, lorsqu'une valeur d'âge est reçue, elle DOIT être interprétée par rapport à l'heure à laquelle la demande a été initiée, et non l'heure à laquelle la réponse a été reçue. Cet algorithme donne un comportement prudent quel que soit le délai rencontré. Ainsi, on calcule :

corrected_initial_age = corrected_received_age + (maintenant - heure_de_demande)

où "request_time" est l'heure (conformément à l'horloge locale) où la demande qui a provoquée cette réponse a été envoyée.

Résumé de l'algorithme de calcul de l'heure, lorsqu'une antémémoire reçoit une réponse :

```
  /*
  * âge_valeur
  *  est la valeur de Age : l'en-tête reçu par l'antémémoire avec cette réponse.
  * date_valeur
  *  est la valeur de l'en-tête Date du serveur d'origine
   * heure_de_demande
  *  est l'heure (locale) à laquelle l'antémémoire a fait la demande qui donne
cette réponse cachée
  * heure_de_réponse
  *  est l'heure (locale) à laquelle l'antémémoire a reçu la réponse
  * maintenant
  *  est l'heure actuelle (locale)
  */
```

```
apparent_âge = max(0, heure_de_réponse - date_valeur);
corrected_received_age = max(apparent_age, âge_valeur);
response_delay = heure_de_réponse - heure_de_demande;
corrected_initial_age = corrected_received_age + response_delay;
resident_time = maintenant - heure_de_réponse;
current_age   = corrected_initial_age + resident_time;
```

L'âge actuel current_age d'une entrée d'antémémoire est calculé en ajoutant la durée (en secondes) depuis que l'entrée d'antémémoire a été validée pour la dernière fois par le serveur d'origine au corrected_initial_age. Lorsqu'une réponse est générée à partir d'une entrée d'antémémoire, l'antémémoire DOIT inclure un seul champ d'en-tête Age dans la réponse avec une valeur égale au current_age de l'entrée d'antémémoire.

La présence d'un champ d'en-tête Age dans une réponse implique qu'une réponse n'est pas de première main. Cependant, l'inverse n'est pas vrai, car l'absence d'un champ d'en-tête Age dans une réponse n'implique pas que la réponse soit de première main, sauf si toutes les antémémoires le long du chemin de la demande sont conformes à HTTP/1.1 (c'est-à-dire que les plus anciennes antémémoires HTTP ne mettent pas en œuvre le champ d'en-tête Age).

13.2.4 Calculs d'expiration

Pour décider si une réponse est fraîche ou périmée, on a besoin de comparer sa durée de vie de fraîcheur à son âge. L'âge est calculé comme décrit au paragraphe 13.2.3 ; ici, on décrit comment calculer la durée de vie de fraîcheur, et comment déterminer si une réponse a expiré. Dans l'exposé ci-dessous, les valeurs peuvent être représentées sous toute forme appropriée pour les opérations arithmétiques.

On utilise le terme "expires_value" pour noter la valeur de l'en-tête Expires. On utilise le terme "max_âge_valeur" pour noter une valeur appropriée du nombre de secondes porté par la directive "max-âge" de l'en-tête Cache-Control dans une réponse (voir au paragraphe 14.9.3).

La directive max-âge prend le pas sur Expires, de sorte que si max-âge est présent dans une réponse, le calcul est simplement :

```
freshness_lifetime = max_âge_valeur
```

Autrement, si Expires est présent dans la réponse, le calcul est :

```
freshness_lifetime = expires_value - date_valeur
```

Noter qu'aucun de ces calculs n'est vulnérable au biais d'horloge, car toutes les informations viennent du serveur d'origine.

Si ni Expires, ni Cache-Control: max-âge, ni Cache-Control: s-maxage (voir au paragraphe 14.9.3) n'apparaissent dans la réponse, et si la réponse n'inclut pas d'autre restriction sur la mise en antémémoire, l'antémémoire PEUT calculer une durée de vie de fraîcheur en utilisant une méthode heuristique. L'antémémoire DOIT attacher l'avertissement 113 à toute réponse dont l'âge est de plus de 24 heures si un tel avertissement n'a déjà été ajouté.

Aussi, si la réponse n'a pas une heure Last-Modified, la valeur d'expiration heuristique DEVRAIT être inférieure ou égale à une fraction de l'intervalle depuis cette heure. Un réglage normal de cette fraction pourrait être 10 %.

Le calcul pour déterminer si une réponse a expiré est assez simple :

```
réponse_fraîche = (durée_de_vie_de_fraîcheur > âge_actuel)
```

13.2.5 Réduction de l'ambiguïté des valeurs d'expiration

Comme les valeurs d'expiration sont allouées de façon optimiste, il est possible que deux antémémoires contiennent des valeurs fraîches qui soient différentes pour la même ressource.

Si un client qui effectue une restitution reçoit une réponse qui n'est pas de première main pour une demande

qui était déjà fraîche dans sa propre antémémoire, et si l'en-tête Date dans son entrée d'antémémoire existante est plus récente que la Date de la nouvelle réponse, le client PEUT alors ignorer la réponse. S'il en est ainsi, il PEUT réessayer la demande avec une directive "Cache-Control: max-âge=0" (voir au paragraphe 14.9) pour forcer à une vérification avec le serveur d'origine. Si une antémémoire a deux réponses fraîches pour la même représentation avec des valideurs différents, elle DOIT utiliser celle avec l'en-tête de date le plus récent. Cette situation pourrait survenir parce que l'antémémoire rassemble les réponses provenant d'autres antémémoires, ou parce qu'un client a demandé un rechargement ou une revalidation d'une entrée d'antémémoire apparemment fraîche.

13.2.6 Réduction de l'ambiguïté des réponses multiples

Comme un client peut recevoir des réponses via plusieurs chemins, certaines réponses s'écoulent à travers un ensemble d'antémémoires et d'autres réponses s'écoulent à travers un ensemble différent d'antémémoires, un client pourrait recevoir des réponses dans un ordre différent de celui dans lequel le serveur d'origine les a envoyées. Il serait souhaitable que le client utilise la réponse générée le plus récemment, même si des réponses plus anciennes sont apparemment encore fraîches.

Ni l'étiquette d'entité ni la valeur d'expiration ne peuvent imposer un ordre des réponses, car il est possible que la dernière réponse porte intentionnellement une heure d'expiration plus récente. Les valeurs de Date sont ordonnées avec une granularité d'une seconde.

Lorsque un client essaye de revalider une entrée d'antémémoire, et que la réponse qu'il reçoit contient un en-tête Date qui paraît plus ancien que celui de l'entrée existante, le client DEVRAIT alors répéter la demande sans condition, et inclure Cache-Control: max-âge=0 pour forcer toute antémémoire intermédiaire à valider directement sa copie avec le serveur d'origine, ou Cache-Control: no-cache pour forcer toute antémémoire intermédiaire à obtenir une nouvelle copie du serveur d'origine.

Si les valeurs de Date sont égales, le client PEUT alors utiliser une des réponses (ou PEUT, si il est extrêmement prudent, demander une nouvelle réponse). Les serveurs NE DOIVENT PAS dépendre des clients pour leur capacité à choisir de façon déterministe entre les réponses générées durant la même seconde, si leurs heures d'expiration se recouvrent.

13.3 Modèle de validation

Lorsqu'une antémémoire a une entrée périmée qu'elle voudrait utiliser comme réponse à la demande d'un client, elle doit d'abord vérifier avec le serveur d'origine (ou éventuellement une antémémoire intermédiaire avec une réponse fraîche) pour voir si son entrée d'antémémoire est toujours utilisable. On appelle cela "valider" l'entrée d'antémémoire. Comme on ne veut pas supporter la redondance de la retransmission de la réponse complète si l'entrée de l'antémémoire est bonne, et si on ne veut pas supporter la redondance d'un aller retour supplémentaire si l'entrée d'antémémoire est invalide, le protocole HTTP/1.1 prend en charge l'utilisation de méthodes conditionnelles.

Les caractéristiques clé du protocole de prise en charge des méthodes conditionnelles sont celles qui sont concernées par les "valideurs d'antémémoire". Lorsqu'un serveur d'origine génère une réponse complète, il y attache des sortes de valideurs, qui sont conservés avec l'entrée d'antémémoire. Lorsqu'un client (antémémoire d'agent d'utilisateur ou de mandataire) fait une demande conditionnelle pour une ressource pour laquelle il a une entrée d'antémémoire, il inclut le valideur associé dans la demande.

Le serveur vérifie alors ce valideur par rapport au valideur en cours pour l'entité, et, si ils correspondent (voir au paragraphe 13.3.3), il répond par un code d'état particulier (habituellement, 304 (Non modifié)) et pas de corps d'entité. Autrement, il retourne une réponse complète (y compris le corps d'entité). Et donc, on évite de transmettre la réponse complète si les valideurs correspondent, et on évite un aller-retour supplémentaire si ils ne correspondent pas.

Dans HTTP/1.1, une demande conditionnelle ressemble exactement à la même chose qu'une demande normale pour la même ressource, excepté qu'elle porte un en-tête particulier (qui inclut le valideur) qui transforme implicitement la méthode (normalement, GET) en un conditionnel.

Le protocole inclut à la fois des sens positifs et négatifs de conditions de validation d'antémémoire. C'est-à-dire qu'il est possible de demander qu'une méthode soit effectuée si et seulement si un valideur correspond ou si et

seulement si aucun valideur ne correspond.

Note : Une réponse qui n'a pas de valideur peut quand même être en antémémoire, et servie à partir d'une antémémoire jusqu'à ce qu'elle arrive à expiration, à moins que cela ne soit explicitement interdit par une directive de contrôle d'antémémoire. Cependant, une antémémoire ne peut pas faire une restitution conditionnelle si elle n'a pas de valideur pour l'entité, ce qui signifie qu'elle ne pourra pas être rafraîchie après son expiration.

13.3.1 Dernières dates modifiées

La valeur de champ d'en-tête d'entité Last-Modified est souvent utilisée comme valideur d'antémémoire. En termes simples, une entrée d'antémémoire est considérée valide si l'entité n'a pas été modifiée depuis la valeur Last-Modified.

13.3.2 Valideurs d'antémémoire d'étiquette d'entité

La valeur de champ d'en-tête de réponse ETag, une étiquette d'entité, donne un valideur d'antémémoire "opaque". Elle peut permettre une validation plus fiable dans des situations où il n'est pas pratique de mémoriser des dates de modification, lorsque la résolution d'une seconde des valeurs de date de HTTP n'est pas suffisante, ou lorsque le serveur d'origine souhaite éviter certains paradoxes qui pourraient découler de l'utilisation de dates de modification.

Les étiquettes d'entité sont décrites au paragraphe 3.11. Les en-têtes utilisés avec les étiquettes d'entité sont décrits aux paragraphes 14.19, 14.24, 14.26 et 14.44.

13.3.3 Valideurs forts et faibles

Comme à la fois les serveurs d'origine et les antémémoires vont comparer deux valideurs pour décider si ils représentent la même entité ou des entités différentes, on s'attendrait normalement à ce que si l'entité (le corps d'entité ou tout en-tête d'entité) change d'une façon quelconque, le valideur associé devrait changer aussi. Si c'est vrai, on dit alors que ce valideur est un "valideur fort".

Cependant, il peut y avoir des cas où un serveur préfère ne changer le valideur que sur des changements sémantiquement significatifs, et non lorsque change un aspect insignifiant de l'entité. Un valideur qui ne change pas toujours lorsque la ressource change est un "valideur faible". Les étiquettes d'entités sont normalement des "valideurs forts", mais le protocole fournit un mécanisme pour étiqueter une étiquette d'entité comme "faible". On peut penser qu'est un valideur fort celui qui change chaque fois que changent les bits d'une entité, alors qu'une valeur faible change chaque fois que la signification d'une entité change. Autrement, on peut penser qu'un valideur fort fait partie d'un identifiant d'une entité spécifique, alors qu'un validateur faible fait partie d'un identifiant pour un ensemble d'entités sémantiquement équivalentes.

Note : Un exemple de valideur fort est un entier qui est incrémenté dans une mémorisation stable chaque fois qu'une entité est changée.

L'heure de modification d'une entité, si elle est représentée avec une résolution d'une seconde, pourrait être un valideur faible, car il est possible que la ressource soit modifiée deux fois dans une seule seconde.

La prise en charge des valideurs faibles est facultative. Cependant, les valideurs faibles permettent une mise en antémémoire plus efficace des objets équivalents ; par exemple, un compteur sur un site est probablement assez bon s'il est mis à jour tous les jours ou semaines, et toute valeur durant cette période sera vraisemblablement "assez bonne" pour être équivalente.

Une "utilisation" d'un valideur est quand un client génère une demande et inclut le valideur dans un champ d'en-tête de validation, ou quand un serveur compare deux valideurs.

Les valideurs forts sont utilisables dans tout contexte. Les valideurs faibles ne sont utilisables que dans des contextes qui ne dépendent pas de l'égalité exacte d'une entité. Par exemple, les deux sortes sont utilisables pour un GET conditionnel d'une entité complète. Cependant, seul un valideur fort est utilisable pour une restitution de sous-gamme, car autrement, le client pourrait finir avec une entité incohérente en interne.

Les clients PEUVENT produire des demandes GET simples (non de sous-gamme) avec des valideurs faibles ou forts. Les clients NE DOIVENT PAS utiliser des valideurs faibles dans d'autres formes de demande.

La seule fonction que définisse le protocole HTTP/1.1 sur les valideurs est la comparaison. Il y a deux fonctions de comparaison de valideur, selon que le contexte de comparaison permet l'utilisation ou non de valideurs faibles :

- La fonction de comparaison forte : afin d'être considérés comme égaux, les deux valideurs DOIVENT être identiques de toutes les façons, et les deux NE DOIVENT PAS être faibles.
- La fonction de comparaison faible : afin d'être considérés comme égaux, les deux valideurs DOIVENT être identiques de toutes les façons, mais l'un ou l'autre ou les deux PEUVENT être marqués comme "faibles" sans affecter le résultat.

Une étiquette d'entité est forte sauf mention expresse comme faible. Le paragraphe 3.11 donne la syntaxe des étiquettes d'entité.

Une heure Last-Modified, lorsqu'elle est utilisée comme un valideur dans une demande, est implicitement faible sauf s'il est possible de déduire qu'elle est forte, en utilisant les règles suivantes :

- le valideur est à comparer par un serveur d'origine au valideur en cours réel pour l'entité et,
- le serveur d'origine doit savoir de façon fiable que l'entité associée n'a pas changé deux fois durant la seconde couverte par le valideur présenté.
  ou
- Le valideur est sur le point d'être utilisé par un client dans un en-tête If-Modified-Since ou If-Unmodified-Since, parce que le client a une entrée d'antémémoire pour l'entité associée, et
- cette entrée d'antémémoire inclut une valeur Date, qui donne l'heure à laquelle le serveur d'origine a envoyé la réponse d'origine, et
- l'heure Last-Modified présentée est au moins 60 secondes avant la valeur de Date.
  ou
- Le valideur va être comparé par une antémémoire intermédiaire au valideur mémorisé dans son entrée d'antémémoire pour l'entité, et
- cette entrée d'antémémoire inclut une valeur Date, qui donne l'heure à laquelle le serveur d'origine a envoyé la réponse d'origine, et
- l'heure Last-Modified présentée est au moins 60 secondes avant la valeur de Date.

Cette méthode s'appuie sur le fait que si deux réponses différentes sont envoyées par le serveur d'origine durant la même seconde, mais que les deux ont la même heure Last-Modified, au moins une de ces réponses aurait alors une valeur de Date égale à son heure Last-Modified. La limite arbitraire de 60 secondes préserve de la possibilité que les valeurs de Date et de Last-Modified soient générées à partir d'horloges différentes, ou à des heures différentes durant la préparation de la réponse. Une mise en œuvre PEUT utiliser une valeur supérieure à 60 secondes, si on croit que 60 secondes est trop court.

Si un client souhaite effectuer une restitution de sous-gamme sur une valeur pour laquelle il a seulement une heure Last-Modified et pas de valideur opaque, il ne PEUT le faire que si l'heure Last-Modified est forte, dans le sens décrit ici.

Une antémémoire ou un serveur d'origine recevant une demande conditionnelle, autre qu'une demande GET à corps complet, DOIT utiliser la fonction de comparaison forte pour évaluer la condition.

Ces règles permettent aux antémémoires et aux clients HTTP/1.1 d'effectuer en toute sécurité des restitutions de sous-gamme sur des valeurs qui ont été obtenues de serveurs HTTP/1.0.

13.3.4 Règles sur le moment où utiliser les étiquettes d'entité et les dates Last-Modified

On a adopté un ensemble de règles et recommandations pour les serveurs d'origine, clients, et antémémoires concernant le moment où les divers types de valideur devraient être utilisés et pour quel objet.

Les serveurs d'origine HTTP/1.1 :

- DEVRAIENT envoyer un valideur d'étiquette d'entité sauf s'il n'est pas faisable d'en générer un.
- PEUVENT envoyer une étiquette d'entité faible à la place d'une étiquette d'entité forte, si les

considérations de performances acceptent l'utilisation d'étiquettes d'entité faibles, ou si il n'est pas faisable d'envoyer une étiquette d'entité forte.
- DEVRAIENT envoyer une valeur Last-Modified si il est faisable d'en envoyer une, sauf si le risque d'une rupture de la transparence sémantique pouvant résulter de l'utilisation de cette date dans un en-tête If-Modified-Since conduirait à de sérieux problèmes.

En d'autres termes, le comportement préféré pour un serveur d'origine HTTP/1.1 est d'envoyer à la fois une étiquette d'entité forte et une valeur Last-Modified.

Pour être légale, une étiquette d'entité forte DOIT changer chaque fois que change d'une façon quelconque la valeur de l'entité associée. Une étiquette d'entité faible DEVRAIT changer chaque fois que l'entité associée change d'une façon sémantiquement significative.

Note : Afin de fournir une mise en antémémoire sémantiquement transparente, un serveur d'origine doit éviter de réutiliser une valeur spécifique d'étiquette d'entité forte pour deux entités différentes, ou de réutiliser une valeur spécifique d'étiquette d'entité faible pour deux entités sémantiquement différentes. Les entrées d'antémémoire peuvent durer sur des périodes arbitrairement longues, sans considération des heures d'expiration, aussi il peut être inapproprié de s'attendre à ce qu'une antémémoire n'essaye jamais de valider une entrée en utilisant un valideur qu'elle a obtenu dans un passé plus ou moins lointain.

Le client HTTP/1.1 :
- Si une étiquette d'entité a été fournie par le serveur d'origine, il DOIT utiliser cette étiquette d'entité dans toute demande conditionnelle d'antémémoire (en utilisant If-Match ou If-None-Match).
- Si seule une valeur Last-Modified a été fournie par le serveur d'origine, il DEVRAIT utiliser cette valeur dans des demandes conditionnelles d'antémémoire qui ne sont pas de sous-gamme (utilisant If-Modified-Since).
- Si seule une valeur Last-Modified a été fournie par un serveur d'origine HTTP/1.0, il PEUT utiliser cette valeur dans des demandes conditionnelles d'antémémoire de sous-gamme (utilisant If-Unmodified-Since:). L'agent d'utilisateur DEVRAIT fournir un moyen pour le désactiver, en cas de difficultés.
- Si une étiquette d'entité et une valeur Last-Modified ont toutes deux été fournies par le serveur d'origine, il DEVRAIT utiliser les deux valideurs dans des demandes d'antémémoire conditionnelles. Ceci permet à la fois à des antémémoires HTTP/1.0 et HTTP/1.1 de répondre de façon appropriée.

Un serveur d'origine HTTP/1.1, à réception d'une demande conditionnelle qui inclut à la fois une date Last-Modified (par exemple, dans un champ d'en-tête If-Modified-Since ou If-Unmodified-Since) et une ou plusieurs étiquettes d'entité (par exemple, dans un champ d'en-tête If-Match, If-None-Match, ou If-Range) comme valideurs d'antémémoire, NE DOIT PAS retourner une réponse d'état de 304 (Non modifié) si ce n'est pas cohérent avec tous les champs d'en-tête conditionnels dans la demande.

Un mandataire de mise en antémémoire HTTP/1.1, à réception d'une demande conditionnelle qui inclut à la fois une date Last-Modified et une ou plusieurs étiquette d'entités comme valideurs d'antémémoire, NE DOIT PAS retourner une réponse mise localement en antémémoire au client à moins que cette réponse d'antémémoire ne soit cohérente avec tous les champs d'en-tête conditionnels dans la demande.

Note : Le principe général de ces règles est que les serveurs et clients HTTP/1.1 devraient transmettre autant d'informations non redondantes qu'il en est de disponible dans leurs réponses et demandes. Les systèmes HTTP/1.1 qui reçoivent ces informations feront les hypothèses les plus prudentes sur les valideurs qu'ils reçoivent.

Les clients et antémémoires HTTP/1.0 ignoreront les étiquettes d'entité. Généralement, les dernières valeurs modifiées reçues ou utilisées par ces systèmes accepteront les mises en antémémoire transparentes et efficaces, et donc les serveurs d'origine HTTP/1.1 devraient fournir les valeurs Last-Modified. Dans les rares cas où l'utilisation d'une valeur Last-Modified comme valideur par un système HTTP/1.0 pourrait résulter en un sérieux problème, les serveurs d'origine HTTP/1.1 devraient ne pas la fournir.

13.3.5 Conditions de non validation

Le principe qui sous-tend les étiquettes d'entité est que seul l'auteur du service connaît assez bien la sémantique d'une ressource pour choisir un mécanisme approprié de validation d'antémémoire, et la spécification de toute fonction de comparaison de valideur plus complexe qu'une égalité d'octet ouvrirait une

boîte de Pandore. Et donc, les comparaisons de tout autre en-tête (excepté Last-Modified, pour la compatibilité avec HTTP/1.0) ne sont jamais utilisées pour les besoins de la validation d'une entrée d'antémémoire.

13.4 Conditions de mise en antémémoire de la réponse

Sauf si c'est spécifiquement exigé par une directive de contrôle d'antémémoire (paragraphe 14.9), un système de mise en antémémoire PEUT toujours mémoriser une réponse de succès (voir au paragraphe 13.8) comme entrée d'antémémoire, PEUT la retourner sans validation si elle est fraîche, et PEUT la retourner après une validation réussie. Si il n'y a ni valideur d'antémémoire, ni un temps d'expiration explicite associé à une réponse, on ne doit pas s'attendre à ce qu'elle soit mise en antémémoire, mais certaines antémémoires PEUVENT violer cette attente (par exemple, lorsqu'il n'y a que peu ou pas du tout de connexité réseau). Un client peut habituellement détecter que de telles réponses ont été tirées d'une antémémoire en comparant l'en-tête de Date à l'heure actuelle.

Note : Certaines antémémoires HTTP/1.0 sont réputées violer cette attente sans fournir aucun avertissement.

Cependant, dans certains cas, il peut être inapproprié pour une antémémoire de conserver une entité, ou de la retourner en réponse à une demande ultérieure. Cela pourrait être parce que une transparence sémantique absolue est jugée nécessaire par l'auteur du service, ou pour des considérations de sécurité ou de confidentialité. Certaines directives de contrôle d'antémémoire sont donc fournies de telle sorte que le serveur puisse indiquer que certaines entités de ressource, ou des portions de celles-ci, ne sont pas à mettre en antémémoire, indépendamment de toute autre considération.

Noter que le paragraphe 14.8 empêche normalement une antémémoire partagée de sauvegarder et retourner une réponse à une demande précédente si cette demande comporte un en-tête Authorization.

Une réponse reçue avec un code d'état de 200, 203, 206, 300, 301 ou 410 PEUT être mémorisée par une antémémoire et utilisée en réponse à une demande ultérieure, sous réserve du mécanisme d'expiration, sauf si une directive de contrôle d'antémémoire interdit la mise en antémémoire. Cependant, une antémémoire qui n'accepte pas les en-têtes Range et Content-Range NE DOIT PAS mettre en antémémoire les réponses 206 (Contenu partiel).

Une réponse reçue avec tout autre code d'état (par exemple les codes d'état 302 et 307) NE DOIT PAS être retournée dans une réponse à une demande ultérieure à moins que des directives de contrôle d'antémémoire ou un ou d'autres en-têtes ne le permettent explicitement. Par exemple, cela inclut ce qui suit : un en-tête Expires (paragraphe 14.21) ; une directive de contrôle d'antémémoire "max-âge", "s-maxage", "must-revalidate", "mandataire-revalidate", "public" ou "private" (paragraphe 14.9).

13.5 Construction des réponses à partir des antémémoires

L'objet d'une antémémoire HTTP est de mémoriser les informations reçues en réponse à des demandes pour les utiliser à répondre à des demandes futures. Dans de nombreux cas, une antémémoire retourne simplement la partie appropriée d'une réponse au demandeur. Cependant, si l'antémémoire détient une entrée d'antémémoire qui se fonde sur une réponse précédente, elle peut avoir à combiner des parties d'une nouvelle réponse avec ce qui est détenu dans l'entrée d'antémémoire.

13.5.1 En-têtes de bout en bout et bond par bond

Pour les besoins de la définition du comportement des antémémoires et des mandataires qui ne mettent pas en antémémoire, on divise les en-têtes HTTP en deux catégories :

- Les en-têtes de bout en bout, qui sont transmis à l'ultime receveur d'une demande ou réponse. Les en-têtes de bout en bout dans les réponses DOIVENT être mémorisés au titre d'une entrée d'antémémoire et DOIVENT être transmis dans toute réponse formée à partir d'une entrée d'antémémoire.
- Les en-têtes de bond par bond, qui n'ont de signification que pour une seule connexion de niveau transport, et ne sont pas mémorisés par les antémémoiress ou transmis par les mandataires.

Les en-têtes HTTP/1.1 qui suivent sont des en-têtes bond par bond :

- Connection (connexion)
- Keep-Alive (garder en vie)

- Proxy-Authenticate (authentification du mandataire)
- Proxy-Authorization (autorisation du mandataire)
- TE (Transfert d'extension)
- Trailers (en-queues)
- Transfer-Encoding (codage de transfert)
- Upgrade (mise à jour)

Tous les autres en-têtes définis par HTTP/1.1 sont des en-têtes de bout en bout.

La liste des autres en-têtes bond par bond DOIT figurer dans un en-tête Connection (paragraphe 14.10) pour être introduite dans HTTP/1.1 (ou une version ultérieure).

13.5.2 En-têtes non modifiables

Certaines caractéristiques du protocole HTTP/1.1, telles que Digest Authentication (authentification par résumé), dépendent de la valeur de certains en-têtes de bout en bout. Un mandataire transparent NE DEVRAIT PAS modifier un en-tête de bout en bout si la définition de cet en-tête ne l'exige pas ou ne le permet pas spécifiquement.
Un mandataire transparent NE DOIT modifier aucun des champs suivants dans une demande ou réponse, et il NE DOIT ajouter aucun de ces champs si ils ne sont pas déjà présents :

- Content-Location (localisation de contenu)
- Content-MD5 (contenu MD5)
- ETag (étiquette d'entité)
- Last-Modified (dernière modification)

Un mandataire transparent NE DOIT modifier aucun des champs suivants dans une réponse :

- Expires

mais il PEUT ajouter un de ces champs s'ils ne sont pas déjà présents. Si un en-tête Expires est ajouté, il DOIT lui être donné une valeur de champ identique à celle de l'en-tête Date dans cette réponse.

Un mandataire NE DOIT modifier ou ajouter aucun des champs suivants dans un message qui contient la directive de contrôle d'antémémoire pas de transformation, ou dans toute demande :

- Content-Encoding (codage de contenu)
- Content-Range (gamme de contenu)
- Content-Type (type de contenu)

Un mandataire non transparent PEUT modifier ou ajouter ces champs à un message qui ne comporte pas de directive no-transform, mais s'il le fait, il DOIT ajouter un avertissement 214 (Transformation appliquée) s'il n'en apparaît pas déjà un dans le message (voir au paragraphe 14.46).

Avertissement : une modification inutile des en-têtes de bout en bout peut causer des défaillances de l'authentification si des mécanismes d'authentification plus forts sont introduits dans des versions ultérieures de HTTP. De tels mécanismes d'authentification PEUVENT s'appuyer sur des valeurs de champs d'en-tête dont la liste ne figure pas ici.

Le champ Content-Length (longueur du contenu) d'une demande ou réponse est ajouté ou supprimé selon les règles du paragraphe 4.4. Un mandataire transparent DOIT préserver la longueur d'entité (paragraphe 7.2.2) du corps de l'entité, bien qu'il PUISSE changer la longueur de transfert (paragraphe 4.4).

13.5.3 Combinaison d'en-têtes

Lorsqu'une antémémoire fait une demande de validation à un serveur, et que le serveur fournit une réponse 304 (Non modifiée) ou une réponse 206 (Contenu partiel), l'antémémoire construit alors une réponse à envoyer au client demandeur.

Si le code d'état est 304 (Non modifié), l'antémémoire utilise le corps d'entité mémorisé dans l'entrée d'antémémoire comme corps d'entité de cette réponse sortante. Si le code d'état est 206 (Contenu partiel) et si les en-têtes ETag ou Last-Modified correspondent exactement, l'antémémoire PEUT combiner les contenus mémorisés dans l'entrée d'antémémoire avec les nouveaux contenus reçus dans la réponse et utiliser le

résultat comme corps d'entité de cette réponse sortante, (voir au paragraphe 13.5.4).

Les en-têtes de bout en bout mémorisés dans l'entrée d'antémémoire sont utilisés pour la réponse construite, excepté que

- tout en-tête Warning mémorisé avec le code d'avertissement 1xx (voir au paragraphe 14.46) DOIT être supprimé de l'entrée d'antémémoire et de la réponse transmise.
- tout en-tête Warning mémorisé avec le code d'avertissement 2xx DOIT être retenu dans l'entrée d'antémémoire et la réponse transmise.
- tout en-tête de bout en bout fourni dans les réponses 304 ou 206 DOIT remplacer les en-têtes correspondants de l'entrée d'antémémoire.

Sauf si l'antémémoire décide de retirer l'entrée d'antémémoire, elle DOIT aussi remplacer les en-têtes de bout en bout mémorisés avec l'entrée d'antémémoire par les en-têtes correspondants reçus dans la réponse entrante, excepté pour les en-têtes Warning comme décrit immédiatement ci-dessus. Si un nom de champ d'en-tête dans la réponse entrante correspond à plus d'un en-tête dans l'entrée d'antémémoire, tous ces vieux en-têtes DOIVENT être remplacés.

En d'autres termes, l'ensemble des en-têtes de bout en bout reçus dans la réponse entrante subrogent tous les en-têtes de bout en bout correspondants mémorisée dans l'entrée d'antémémoire (excepté pour les en-têtes Warning portant le code d'avertissement 1xx, qui sont supprimés même s'ils ne sont pas subrogés).

Note : Cette règle permet à un serveur d'origine d'utiliser une réponse 304 (Non modifié) ou 206 (Contenu partiel) pour mettre à jour tout en-tête associé à une réponse précédente pour la même entité ou sous-gamme, bien qu'il ne soit pas toujours significatif ou correct de le faire. La présente règle ne permet pas à un serveur d'origine d'utiliser une réponse 304 (Non modifié) ou 206 (Contenu partiel) pour supprimer entièrement un en-tête qu'il avait fourni avec une réponse précédente.

13.5.4 Combinaison des gammes d'octet

Une réponse ne peut transférer qu'une sous-gamme des octets d'un corps d'entité, soit parce que la demande inclut une ou plusieurs spécifications de gamme, soit parce qu'une connexion a été rompue prématurément. Après plusieurs transferts de cette sorte, une antémémoire peut avoir reçu plusieurs gammes du même corps d'entité.

Si une antémémoire a mémorisé un ensemble non vide de sous-gammes pour une entité, et qu'une réponse entrante transfère une autre sous-gamme, l'antémémoire PEUT combiner la nouvelle sous-gamme avec l'ensemble existant si les deux conditions suivantes sont réunies :

- La réponse entrante et l'entrée d'antémémoire ont toutes deux un valideur d'antémémoire.
- Les deux valideurs d'antémémoire correspondent en utilisant la fonction de comparaison forte (voir au paragraphe 13.3.3).

Si aucune des exigences n'est satisfaite, l'antémémoire DOIT n'utiliser que la réponse partielle la plus récente (sur la base des valeurs de Date transmises avec chaque réponse, et en utilisant la réponse entrante si ces valeurs sont égales ou manquantes), et DOIT éliminer les autres informations partielles.

13.6 Mise en antémémoire des réponses négociées

L'utilisation de la négociation de contenu conduite par le serveur (paragraphe 12.1), telle qu'indiquée par la présence d'un champ d'en-tête Vary dans une réponse, altère les conditions et procédures par lesquelles une antémémoire peut utiliser la réponse pour des demandes ultérieures. Voir au paragraphe 14.44 l'utilisation du champ d'en-tête Vary par les serveurs.

Un serveur DEVRAIT utiliser le champ d'en-tête Vary pour informer une antémémoire des champs d'en-tête de demande qui ont été utilisés pour choisir parmi les multiples représentations d'une réponse inscriptible en antémémoire soumise à la négociation conduite par le serveur. L'ensemble des champs d'en-tête nommés par la valeur du champ Vary est appelé les en-têtes de demande "sélecteurs".

Lorsque l'antémémoire reçoit une demande ultérieure dont l'URI de demande spécifie une ou plusieurs entrées d'antémémoire comprenant un champ d'en-tête Vary, l'antémémoire NE DOIT PAS utiliser une telle entrée d'antémémoire pour construire une réponse à la nouvelle demande si tous les en-têtes de demande sélecteurs

présents dans la nouvelle demande ne correspondent pas aux en-têtes de demande correspondants mémorisés dans la demande d'origine.

Les en-têtes de demande sélecteurs provenant des deux demandes ne sont définis comme correspondants que si et seulement si les en-têtes de demande sélecteurs de la première demande peuvent être transformés en en-têtes de demande sélecteurs dans la seconde demande en ajoutant ou retranchant des espaces linéaires (LWS) aux places où c'est admis par le BNF correspondant, et/ou en combinant plusieurs champs d'en-tête de message avec le même nom de champ en suivant les règles sur les en-têtes de message du paragraphe 4.2.

Une valeur de champ d'en-tête Vary de "*" échoue toujours à correspondre et les demandes suivantes sur cette ressource ne peuvent être interprétées correctement par le serveur d'origine.

Si les champs d'en-tête de demande sélecteurs pour l'entrée d'antémémoire ne correspondent pas aux champs d'en-tête de demande sélecteurs de la nouvelle demande, l'antémémoire NE DOIT alors PAS utiliser d'entrée d'antémémoire pour satisfaire la demande sauf si elle assure le premier relais de la nouvelle demande vers le serveur d'origine dans une demande conditionnelle et que serveur répond par 304 (Non modifiée), y compris une étiquette d'entité ou un Content-Location qui indique l'entité à utiliser.

Si une étiquette d'entité a été allouée à une représentation d'antémémoire, la demande transmise DEVRAIT être conditionnelle et inclure les étiquettes de l'entité dans un champ d'en-tête If-None-Match provenant de toutes ses entrées d'antémémoire pour la ressource. Ceci transporte au serveur l'ensemble des entités actuellement détenues par l'antémémoire, de sorte que si une de ces entités correspond à l'entité demandée, le serveur peut utiliser le champ d'en-tête ETag dans sa réponse 304 (Non modifiée) pour dire à l'antémémoire quelle entrée est appropriée. Si l'étiquette d'entité de la nouvelle réponse correspond à celle d'une entrée existante, la nouvelle réponse DEVRAIT être utilisée pour mettre à jour les champs d'en-tête de l'entrée existante, et le résultat DOIT être retourné au client.

Si une des entrées d'antémémoire existantes contient seulement un contenu partiel pour l'entrée associée, son étiquette d'entité NE DEVRAIT PAS être incluse dans le champ d'en-tête If-None-Match sauf si la demande est pour une gamme qui serait complètement satisfaite par cette entrée.

Si une antémémoire reçoit une réponse de succès dont le champ Content-Location correspond à celle d'une entrée d'antémémoire existante pour le même URI de demande, dont l'étiquette d'entité diffère de celle de l'entrée existante, et dont la Date est plus récente que celle de l'entrée existante, l'entrée existante NE DEVRAIT PAS être retournée en réponse à des demandes ultérieures et DEVRAIT être supprimée de l'antémémoire.

13.7 Antémémoires partagées et non partagées

Pour des raisons de sécurité et de confidentialité, il est nécessaire de faire une distinction entre antémémoires "partagées" et "non partagées". Une antémémoire non partagée est celle qui n'est accessible qu'à un seul utilisateur. Dans ce cas, l'accessibilité DEVRAIT être mise en application par les mécanismes de sécurité appropriés. Toutes les autres antémémoires sont considérée comme étant "partagées". D'autres paragraphes de la présente spécification établissent certaines contraintes sur le fonctionnement des antémémoires partagées afin de prévenir les atteintes à la confidentialité ou l'échec des contrôles d'accès.

13.8 Comportement de l'antémémoire en cas d'erreur ou de réponse incomplète

Une antémémoire qui reçoit une réponse incomplète (par exemple, avec moins d'octets de données que spécifié dans un en-tête Content-Length) PEUT mémoriser la réponse. Cependant, l'antémémoire DOIT traiter cela comme une réponse partielle. Les réponses partielles PEUVENT être combinées comme décrit au paragraphe 13.5.4 ; le résultat peut être une réponse complète ou peut encore être partiel. Une antémémoire NE DOIT PAS retourner une réponse partielle à un client sans la marquer explicitement comme telle, en utilisant le code d'état 206 (Contenu partiel). Une antémémoire NE DOIT PAS retourner une réponse partielle utilisant un code d'état de 200 (OK).

Si une antémémoire reçoit une réponse 5xx alors qu'elle essaye de revalider une entrée, elle PEUT soit transmettre cette réponse au client demandeur, soit agir comme si le serveur avait échoué à répondre. Dans ce dernier cas, elle PEUT retourner une réponse précédemment reçue sauf si l'entrée de la mémoire comporte la

directive de commande d'antémémoire "must-revalidate" (voir au paragraphe 14.9).

13.9 Effets collatéraux de GET et HEAD

Sauf si le serveur d'origine interdit explicitement la mise en antémémoire de leurs réponses, l'application des méthodes GET et HEAD à toute ressource NE DEVRAIT PAS avoir d'effets collatéraux qui puissent conduire à un comportement erroné si ces réponses sont tirées d'une antémémoire. Il PEUT toujours y avoir des effets collatéraux, mais une antémémoire n'est pas obligée de considérer de tels effets collatéraux dans ses décisions de mise en antémémoire. Les antémémoires sont toujours supposées observer les restrictions explicites d'un serveur d'origine sur la mise en antémémoire.

On notera une exception à cette règle : comme certaines applications ont traditionnellement utilisé GET et HEAD avec des URL d'interrogation (celles qui contiennent un "?" dans la partie rel_path) pour effectuer des opérations ayant des effets collatéraux significatifs, les antémémoires NE DOIVENT PAS traiter les réponses à de tels URI comme fraîches sauf si le serveur fournit un temps d'expiration explicite. Ceci signifie précisément que les réponses provenant de serveurs HTTP/1.0 pour de tels URI NE DEVRAIENT PAS être tirées d'une antémémoire. Voir au paragraphe 9.1.1 les informations qui s'y rapportent.

13.10 Invalidation après mise à jour ou suppression

Les effets de certaines méthodes effectuées sur une ressource au serveur d'origine peuvent causer la transformation d'une ou plusieurs entrées d'antémémoire existantes en entrées invalides non transparentes. C'est-à-dire que bien qu'elles puissent continuer d'être "fraîches", elles ne reflètent plus avec précision ce que le serveur d'origine retournerait pour une nouvelle demande sur cette ressource.

Il n'y a aucun moyen pour le protocole HTTP de garantir que toutes les entrées d'antémémoire de cette sorte soient marquées comme invalides. Par exemple, la demande qui a causé le changement au serveur d'origine pourrait n'être pas passée à travers le mandataire où une entrée d'antémémoire est mémorisée. Cependant, plusieurs règles aident à réduire la probabilité d'un comportement erroné.

Dans ce paragraphe, la phrase "invalider une entité" signifie que l'antémémoire va soit retirer toutes les instances de cette entité de sa mémoire, soit les marquer comme "invalides" et qu'elles ont besoin d'une revalidation obligatoire avant qu'elles puissent être retournées en réponse à une demande ultérieure.

Certaines méthodes HTTP DOIVENT provoquer l'invalidation d'une entité par une antémémoire. C'est soit l'entité à laquelle se réfère l'URI de demande, soit les en-têtes Location ou Content-Location (s'ils sont présents).

Ces méthodes sont :

- PUT
- DELETE
- POST

Pour prévenir les attaques de déni de service, une invalidation fondée sur l'URI dans un en-tête Location ou Content-Location ne DOIT être effectuée que si la partie hôte est la même que dans l'URI de demande.

Une antémémoire qui voit passer des demandes pour des méthodes qu'elle ne comprend pas DEVRAIT invalider toute entité à laquelle se réfère l'URI de demande.

13.11 Écriture en transfert obligatoire

Toutes les méthodes dont on peut s'attendre à ce qu'elles causent des modifications aux ressources du serveur d'origine DOIVENT être écrites en transfert au serveur d'origine. Ceci inclut actuellement toutes les méthodes excepté GET et HEAD. Une antémémoire NE DOIT PAS répondre à une telle demande de la part d'un client avant d'avoir transmis la demande au serveur entrant, et d'avoir reçu une réponse correspondante de la part du serveur entrant. Ceci n'empêche pas un mandataire d'antémémoire d'envoyer une réponse 100 (Continuer) avant que le serveur entrant n'ait envoyé sa réponse finale.

La solution de remplacement (appelée mise en antémémoire "de recopie" ou "de réécriture") n'est pas permise

dans HTTP/1.1, à cause de la difficulté de fournir des mises à jour cohérentes et des problèmes qui surviennent à cause de défaillances du serveur, de l'antémémoire, ou du réseau avant la réécriture.

13.12 Remplacement d'antémémoire

Si une nouvelle réponse mettable en antémémoire (voir les paragraphes 14.9.2, 13.2.5, 13.2.6 et 13.8) est reçue d'une ressource alors que des réponses existantes pour la même ressource sont en antémémoire, l'antémémoire DEVRAIT utiliser la nouvelle réponse pour répondre à la demande en cours. Elle PEUT l'insérer dans une antémémoire et PEUT, si elle satisfait à toutes les autres exigences, l'utiliser pour répondre à toute demande future qui aurait précédemment causé le retour de la vieille réponse. Si elle insère la nouvelle réponse dans une antémémoire, les règles du paragraphe 13.5.3 s'appliquent.

Note : Une nouvelle réponse qui a une valeur d'en-tête Date plus ancienne que les réponses existantes en antémémoire n'est pas mettable en antémémoire.

13.13 Listes d'historique

Les agents d'utilisateur ont souvent des mécanismes d'historique, tels que des boutons "Back" et des listes d'historique, qui peuvent être utilisées pour afficher une entité récupérée plus tôt dans une session.

Les mécanismes d'historique et les antémémoires sont différents. En particulier, les mécanismes d'historique NE DEVRAIENT PAS essayer de donner une vision sémantiquement transparente de l'état actuel d'une ressource. Un mécanisme d'historique est plutôt destiné à montrer exactement ce que l'utilisateur voyait au moment où la ressource a été restituée.

Par défaut, on n’applique pas de temps d'expiration aux mécanismes d'historique. Si l'entité est toujours mémorisée, un mécanisme d'historique DEVRAIT l'afficher même si l'entité est arrivée à expiration, sauf si l'utilisateur a spécifiquement configuré l'agent pour rafraîchir les documents historiques arrivés à expiration.

Ceci n'est pas destiné à interdire au mécanisme d'historique de dire à l'utilisateur qu'une entité pourrait être périmée.

Note Si les mécanismes de liste d'historique empêchent inutilement les utilisateurs de voir les ressources périmées, cela tendrait à forcer les auteurs de services à éviter d'utiliser les commandes d'expiration et les commandes d'antémémoire HTTP alors qu'ils souhaiteraient le faire. Les auteurs de services peuvent considérer qu'il est important que les utilisateurs ne reçoivent pas de messages d'erreur ou d'avertissement lorsqu'ils utilisent les commandes de navigation (tels que BACK) pour voir les ressources atteintes précédemment. Même si parfois ces ressources ne devraient pas être mises en antémémoire, ou devraient arriver rapidement à expiration, les considérations d'interface d'utilisateur peuvent forcer les auteurs de services à recourir à d'autres moyens d'empêcher la mise en antémémoire (par exemple, les URL "once-only") afin de ne pas subir les effets de mécanismes d'historique qui fonctionnent mal.

14 Définitions des champs d’en-tête

La présente section définit la syntaxe et la sémantique de tous les champs d’en-tête standard de HTTP/1.1. Pour les champs d’en-tête d’entité, l'envoyeur et le receveur se réfèrent tous deux soit au client soit au serveur, selon qui envoie et qui reçoit l'entité.

14.1 Accept

Le champ d'en-tête de demande Accept peut être utilisé pour spécifier certains types de supports qui sont acceptables pour la réponse. Les en-têtes Accept peuvent être utilisés pour indiquer que la demande est spécifiquement limitée à un petit ensemble de types souhaités, comme dans le cas d'une demande d'image en ligne.

```
Accept         = "Accept" ":" #( media-range [ accept-params ] )

media-range    = ( "*/*"
```

```
   | ( type "/" "*" )
  | ( type "/" subtype )
  ) *( ";" parameter )
  accept-params  = ";" "q" "=" qvalue *( accept-extension )
  accept-extension = ";" token [ "=" ( token | quoted-string ) ]
```

Le caractère astérisque "*" est utilisé pour grouper des types de support en gammes, avec "*/*" qui indique tous les types de support et "type/*" qui indique tous les sous-types de ce type. La gamme de support PEUT inclure des paramètres de type de support qui sont applicables à cette gamme.

Chaque gamme de support media-range PEUT être suivie par un ou plusieurs accept-params, commençant par le paramètre "q" pour indiquer un facteur de qualité relative. Le premier paramètre "q" (s'il en est) sépare le ou les paramètres media-range des accept-params. Les facteurs de qualité permettent à l'utilisateur ou agent d'utilisateur d'indiquer le degré de préférence relative pour cette gamme de support, en utilisant l'échelle qvalue de 0 à 1 (paragraphe 3.9). La valeur par défaut est q=1.

Note : L'utilisation du nom de paramètre "q" pour séparer les paramètres de type de support des paramètres d'extension Accept est due à une vieille pratique. Bien que cela empêche d'utiliser tout paramètre de type de support nommé "q" avec une gamme de support, la survenance d'un tel événement est jugée peu probable étant donné l'absence de tout paramètre "q" dans les registres de type de support de l'IANA et l'utilisation rare de tout paramètre de type de support dans Accept. Il est déconseillé d'enregistrer des types futurs de paramètres support nommés "q".

L'exemple <Accept: audio/*; q=0.2, audio/basic> DEVRAIT être interprété comme "Je préfère audio/basic, mais envoie moi tout type audio si c'est le meilleur disponible après un marquage de qualité de 80 %."

Si aucun champ d'en-tête Accept n'est présent, on supposera alors que le client accepte tous les types de support. Si un champ d'en-tête Accept est présent, et si le serveur ne peut pas envoyer une réponse acceptable selon la valeur combinée du champ Accept, le serveur DEVRAIT alors envoyer une réponse 406 (Non acceptable).

Un exemple plus élaboré est :

```
Accept: text/plain; q=0.5, text/html, text/x-dvi; q=0.8, text/x-c
```

Verbalement, ceci serait interprété comme "text/html et text/x-c sont les types de support préférés, mais si ils n'existent pas, envoie l'entité text/x-dvi, et si elle n'existe pas, envoie l'entité text/plain."

Les gammes de support peuvent être subrogées par des gammes de support plus spécifiques ou des types de support spécifiques. Si plus d'une gamme de support s'applique à un type donné, la référence la plus spécifique prend le pas. Par exemple,

```
Accept: text/*, text/html, text/html;level=1, */*
```

a la priorité suivante :

1) text/html;level=1
2) text/html
3) text/*
4) */*

Le facteur de qualité de type de support associé à un type donné est déterminé en trouvant la gamme de support avec la plus haute priorité qui correspond à ce type. Par exemple,

```
Accept: text/*;q=0.3, text/html;q=0.7, text/html;level=1, text/html;level=2;q=0.4,
*/*;q=0.5
```

causerait l'association des valeurs suivantes :

text/html;level=1 = 1
text/html = 0,7
text/plain = 0,3

image/jpeg = 0,5
text/html;level=2 = 0,4
text/html;level=3 = 0,7

Note : Un agent d'utilisateur pourrait être approvisionné avec un ensemble par défaut de valeurs de qualité pour certaines gammes de support. Cependant, sauf si l'agent d'utilisateur est un système fermé qui ne peut pas interagir avec d'autres agents de rapport, cet ensemble par défaut devrait être configurable par l'utilisateur.

14.2 Accept-Charset

Le champ d'en-tête de demande Accept-Charset peut être utilisé pour indiquer quels jeux de caractères sont acceptables pour la réponse. Ce champ permet aux clients capables de comprendre des jeux de caractères plus exhaustifs ou de propos plus particuliers pour signaler cette capacité à un serveur qui est capable de représenter des documents dans ces jeux de caractères.

```
Accept-Charset = "Accept-Charset" ":" 1#( ( charset | "*" )[ ";" "q" "=" qvalue ] )
```

Les valeurs de jeux de caractères sont décrites au paragraphe 3.4. Chaque jeu de caractères (charset) PEUT recevoir une valeur de qualité associée qui représente la préférence de l'utilisateur pour ce charset. La valeur par défaut est q=1. Un exemple est :

```
Accept-Charset: iso-8859-5, unicode-1-1;q=0.8
```

La valeur particulière "*", si elle est présente dans le champ Accept-Charset, correspond à chaque jeu de caractères (y compris ISO-8859-1) qui n'est pas mentionné ailleurs dans le champ Accept-Charset. Si aucun "*" n'est présent dans un champ Accept-Charset, tous les jeux de caractères non explicitement mentionnés obtiennent alors une valeur de qualité de 0, sauf pour ISO-8859-1, qui obtient une valeur de qualité de 1 si elle n'est pas mentionnée explicitement.

Si aucun en-tête Accept-Charset n'est présent, la valeur par défaut est que tout jeu de caractères est acceptable. Si un en-tête Accept-Charset est présent, et si le serveur ne peut pas envoyer une réponse qui soit acceptable selon l'en-tête Accept-Charset, le serveur DEVRAIT alors envoyer une réponse d'erreur avec le code d'état 406 (Non acceptable), quoique l'envoi d'une réponse non acceptable soit aussi permis.

14.3 Accept-Encoding

Le champ d'en-tête de demande Accept-Encoding est similaire à Accept, mais restreint les codages de contenu (paragraphe 3.5) qui sont acceptables dans la réponse.

```
Accept-Encoding  = "Accept-Encoding"":" 1#( codings [ ";" "q" "=" qvalue ] )
codings          = ( content-coding | "*" )
```

Des exemples de son utilisation sont :

```
Accept-Encoding: compress, gzip
Accept-Encoding:
Accept-Encoding: *
Accept-Encoding: compress;q=0.5, gzip;q=1.0
Accept-Encoding: gzip;q=1.0, identity;q=0.5, *;q=0
```

Un serveur vérifie si un codage de contenu est acceptable, selon un champ Accept-Encoding, en utilisant ces règles :

1. Si le codage de contenu est un des codages de contenu dont la liste figure dans le champ Accept-Encoding, il est alors acceptable, sauf s'il est accompagné d'une qvalue de 0. (Comme défini au paragraphe 3.9, une qvalue de 0 signifie "non acceptable".)

2. Le symbole spécial "*" dans un champ Accept-Encoding correspond à tout codage de contenu disponible non explicitement listé dans le champ d'en-tête.

3. Si plusieurs codages de contenu sont acceptables, le codage de contenu acceptable avec la plus haute qvalue différente de zéro est préféré.

4. Le codage de contenu "identity" est toujours acceptable, sauf spécifiquement refusé parce que le champ Accept-Encoding inclut "identity;q=0", ou parce que le champ inclut "*;q=0" et n'inclut pas explicitement le codage de contenu "identity". Si la valeur de champ Accept-Encoding est vide, seul le codage "identity" est alors acceptable.

Si un champ Accept-Encoding est présent dans une demande, et si le serveur ne peut pas envoyer une réponse acceptable selon l'en-tête Accept-Encoding, le serveur DEVRAIT alors envoyer une réponse d'erreur avec le code d'état 406 (Non acceptable).

Si aucun champ Accept-Encoding n'est présent dans une demande, le serveur PEUT supposer que le client acceptera tout codage de contenu. Dans ce cas, si "identity" est un des codages de contenu disponibles, le serveur DEVRAIT alors utiliser le codage de contenu "identity", sauf si il a des informations supplémentaires indiquant qu'un codage de contenu différent serait significatif pour le client.

Note : Si la demande n'inclut pas de champ Accept-Encoding, et si le codage de contenu "identity" n'est pas disponible, les codages de contenu compris habituellement par les clients HTTP/1.0 (c'est-à-dire, "gzip" et "compress") sont préférés ; certains clients plus anciens affichent à tort des messages envoyés avec d'autres codages de contenu. Le serveur pourrait aussi prendre cette décision sur la base d'informations sur l'agent d'utilisateur ou client particulier.

Note : La plupart des applications HTTP/1.0 ne reconnaissent pas ou n'obéissent pas aux qvaleurs associées aux codages de contenu. Cela signifie que les qvaleurs ne vont pas fonctionner et ne sont pas permises avec x-gzip ou x-compress.

14.4 Accept-Language

Le champ d'en-tête de demande Accept-Language est similaire à Accept, mais restreint l'ensemble des langages naturels qui sont préférés comme réponse à la demande. Les étiquettes de langage sont définies au paragraphe 3.10.

```
Accept-Language = "Accept-Language" ":" 1#( language-range [ ";" "q" "=" qvalue ] )
   language-range  = ( ( 1*8ALPHA *( "-" 1*8ALPHA ) ) | "*" )
```

Chaque gamme de langue PEUT recevoir une valeur de qualité associée qui représente une estimation de la préférence de l'utilisateur pour les langages spécifiés par cette gamme. La valeur de qualité par défaut est de "q=1". Par exemple,

```
Accept-Language: da, en-gb;q=0.8, en;q=0.7
```

signifierait : "Je préfère le danois, mais j'accepte l'anglais d'Angleterre et autres types d'anglais." Une gamme de langues correspond à une étiquette de langage si elle est exactement égale à l'étiquette, ou si elle est exactement égale au préfixe de l'étiquette de sorte que le premier caractère de l'étiquette suivant le préfixe soit "-". La gamme particulière "*", si elle est présente dans le champ Accept-Language, correspond à toute étiquette qui ne correspond pas à une autre gamme présente dans le champ Accept-Language.

Note : Cette utilisation d'une règle de correspondance de préfixe n'implique pas que les étiquettes de langue sont allouées aux langages d'une façon telle qu'il soit toujours vrai que si un utilisateur comprend un langage avec une certaine étiquette, cet utilisateur va aussi comprendre toutes les langues ayant les étiquettes dont cette étiquette est un préfixe. La règle du préfixe permet simplement l'utilisation des étiquettes de préfixe si c'est le cas.

Le facteur de qualité de langage alloué à une étiquette de langue par le champ Accept-Language est la valeur de la qualité de la plus longue gamme de langage dans le champ qui correspond à l'étiquette de langage. Si aucune gamme de langue ne correspond à l'étiquette dans le champ, le facteur de qualité de langage alloué est 0. Si aucun en-tête Accept-Language n'est présent dans la demande, le serveur DEVRAIT supposer que tous les langages sont également acceptables. Si un en-tête Accept-Language est présent, toutes les langues auxquelles a été alloué un facteur de qualité supérieur à 0 sont acceptables.

Il pourrait être contraire aux attentes de confidentialité de l'utilisateur d'envoyer un en-tête Accept-Language avec toutes les préférences linguistiques de l'utilisateur dans toutes les demandes. Pour une discussion de

cette question, voir au paragraphe 15.1.4.

Comme l'intelligibilité est étroitement dépendante de l'utilisateur individuel, il est recommandé que les applications client fassent le choix des préférences linguistiques disponibles pour l'utilisateur. Si le choix n'est pas offert, le champ d'en-tête Accept-Language NE DOIT PAS être donné dans la demande.

Note : Quand on offre le choix de la préférence linguistique à l'utilisateur, les développeurs doivent se souvenir que les utilisateurs ne sont pas familiarisés avec les détails des correspondances de langage tels que décrits ci-dessus, et ils devraient fournir les conseils appropriés. À titre d'exemple, les utilisateurs pourraient supposer qu'en choisissant "en-gb", il va leur être servi n'importe quelle sorte d'anglais si l'anglais de Grande Bretagne n'est pas disponible. Un agent d'utilisateur peut suggérer dans un tel cas d'ajouter "en" pour obtenir le meilleur comportement de correspondance.

14.5 Accept-Ranges

Le champ d'en-tête de réponse Accept-Ranges permet au serveur d'indiquer qu'il accepte des gammes de demande pour une ressource :

```
Accept-Ranges     = "Accept-Ranges" ":" acceptable-ranges
   acceptable-ranges = 1#range-unit | "none"
```

Les serveurs d'origine qui acceptent la gamme d'octet des demandes PEUVENT envoyer

```
Accept-Ranges: bytes
```

mais il n'est pas exigé qu'ils le fassent. Les clients PEUVENT générer des demandes de gamme d'octets sans avoir reçu cet en-tête pour la ressource impliquée. Les unités de gamme sont définies au paragraphe 3.12.

Les serveurs qui n'acceptent aucune sorte de demande de gamme pour une ressource PEUVENT envoyer

```
Accept-Ranges: none
```

pour informer le client de ne pas essayer de demande de gamme.

14.6 Age

Le champ d'en-tête de réponse Age convoie l'estimation de l'envoyeur de la quantité de temps écoulé depuis la génération de la réponse (ou de sa revalidation) au serveur d'origine. Une réponse d'antémémoire est "fraîche" si son âge n'excède pas la durée de vie de sa fraîcheur. Les valeurs de Age sont calculées comme spécifié au paragraphe 13.2.3.

```
Age = "Age" ":" âge-valeur
âge-valeur = delta-secondes
```

Les valeurs de Age sont des entiers décimaux non négatifs, représentant le temps en secondes.

Si une antémémoire reçoit une valeur supérieure au plus grand entier positif qu'elle peut représenter, ou si un des calculs d'âge déborde, elle DOIT transmettre un en-tête Age avec une valeur de 2147483648 (2^31). Un serveur HTTP/1.1 qui inclut une antémémoire DOIT inclure un champ d'en-tête Age dans chacune des réponses générées à partir de sa propre antémémoire. Les antémémoires DEVRAIENT utiliser un type arithmétique d'au moins 31 bits de gamme.

14.7 Allow

Le champ d'en-tête d'entité Allow fait la liste de l'ensemble des méthodes prises en charge par la ressource identifiée par l'URI de demande. L'objet de ce champ est strictement d'informer le receveur des méthodes valides associées à la ressource. Un champ d'en-tête Allow DOIT être présent dans une réponse 405 (Méthode non admise).

```
Allow   = "Allow" ":" #Method
```

Exemple d'utilisation :

```
Allow: GET, HEAD, PUT
```

Ce champ ne peut pas empêcher un client d'essayer d'autres méthodes. Cependant, les indications données par la valeur de champ d'en-tête Allow DEVRAIENT être suivies. L'ensemble réel des méthodes admises est défini par le serveur d'origine au moment de chaque demande.

Le champ d'en-tête Allow PEUT être fourni avec une demande PUT pour recommander les méthodes qui devraient être prises en charge par la ressource nouvelle ou modifiée. Le serveur n'est pas obligé de prendre en charge ces méthodes et DEVRAIT inclure un en-tête Allow dans la réponse, donnant les méthodes réellement prises en charge.

Un mandataire NE DOIT PAS modifier le champ d'en-tête Allow même s'il ne comprend pas toutes les méthodes spécifiées, car l'agent d'utilisateur pourrait avoir d'autres moyens de communiquer avec le serveur d'origine.

14.8 Authorization

Un agent d'utilisateur qui souhaite s'authentifier auprès d'un serveur -- habituellement, mais pas nécessairement, après avoir reçu une réponse 401 -- le fait en incluant un champ d'en-tête de demande Authorization avec la demande. La valeur du champ Authorization consiste en accréditifs qui contiennent les informations d'authentification de l'agent d'utilisateur pour le domaine de la ressource demandée.

```
Authorization  = "Authorization" ":" credentials
```

L'authentification d'accès HTTP est décrite dans "Authentification HTTP : Authentification d'accès de base et par résumé" [43]. Si une demande est authentifiée et qu'un domaine est spécifié, les mêmes accréditifs DEVRAIENT être valides pour toutes les autres demandes au sein de ce domaine (en supposant que le schéma d'authentification lui-même n'en exige pas autrement, comme des accréditifs qui varient selon une valeur de confrontation ou qui utilisent des horloges synchronisées).

Lorsque une antémémoire partagée (voir au paragraphe 13.7) reçoit une demande contenant un champ Authorization, elle NE DOIT PAS retourner la réponse correspondante comme une réponse à aucune autre demande, sauf en présence d'une des exceptions spécifiques suivantes :

1. Si la réponse inclut la directive de commande d'antémémoire "s-maxage", l'antémémoire PEUT utiliser cette réponse pour répondre à une demande ultérieure. Mais (si l'âge maximum spécifié est dépassé) un mandataire d'antémémoire DOIT d'abord la revalider avec le serveur d'origine, en utilisant les en-têtes de demande provenant de la nouvelle demande pour permettre au serveur d'origine d'authentifier la nouvelle demande. (Ceci est le comportement défini pour s-maxage.) Si la réponse inclut "s-maxage=0", le mandataire DOIT toujours la revalider avant de la réutiliser.

2. Si la réponse inclut la directive de commande d'antémémoire "must-revalidate", l'antémémoire PEUT utiliser cette réponse en réponse à une demande ultérieure. Mais, si la réponse est périmée, toutes les antémémoires DOIVENT d'abord la revalider auprès du serveur d'origine, en utilisant les en-têtes de demande de la nouvelle demande pour permettre au serveur d'origine d'authentifier la nouvelle demande.

3. Si la réponse inclut la directive de commande d'antémémoire "public", elle PEUT être retournée en réponse à toute demande ultérieure.

14.9 Cache-Control

Le champ d'en-tête général Cache-Control est utilisé pour spécifier des directives qui DOIVENT être respectées par tous les mécanismes de mise en antémémoire le long de la chaîne des demandes/réponses. Les directives spécifient le comportement voulu pour empêcher les antémémoires d'interférer de façon nuisible avec la demande ou la réponse. Ces directives se substituent normalement aux algorithmes de mise en antémémoire par défaut. Les directives d'antémémoire sont unidirectionnelles en ce que la présence d'une directive dans une demande n'implique pas que la même directive soit donnée dans la réponse.

Noter que les antémémoires de HTTP/1.0 peuvent ne pas mettre en œuvre Cache-Control et pourraient ne mettre en œuvre que Pragma: no-cache (voir au paragraphe 14.32).

Les directives d'antémémoire DOIVENT passer par un mandataire ou une passerelle application, indépendamment de leur signification pour cette application, car les directives pourraient être applicables à tous les receveurs le long de la chaîne des demandes/réponses. Il n'est pas possible de spécifier une directive d'antémémoire pour une mémoire spécifique.

```
Cache-Control   = "Cache-Control" ":" 1#cache-directive

cache-directive = cache-request-directive | cache-response-directive

cache-request-directive =
           "no-cache"                          Paragraphe 14.9.1
         | "no-store"                          Paragraphe 14.9.2
         | "max-âge" "=" delta-seconds         Paragraphe 14.9.3, 14.9.4
         | "max-stale" [ "=" delta-seconds ]   Paragraphe 14.9.3
         | "min-fresh" "=" delta-seconds       Paragraphe 14.9.3
         | "no-transform"                      Paragraphe 14.9.5
         | "only-if-cached"                    Paragraphe 14.9.4
         | cache-extension                     Paragraphe 14.9.6

cache-response-directive =
           "public"                                 Paragraphe 14.9.1
         | "private" [ "=" <"> 1#field-name <"> ]   Paragraphe 14.9.1
         | "no-cache" [ "=" <"> 1#field-name <"> ]  Paragraphe 14.9.1
         | "no-store"                               Paragraphe 14.9.2
         | "no-transform"                           Paragraphe 14.9.5
         | "must-revalidate"                        Paragraphe 14.9.4
         | "proxy-revalidate"                       Paragraphe 14.9.4
         | "max-âge" "=" delta-seconds              Paragraphe 14.9.3
         | "s-maxage" "=" delta-seconds             Paragraphe 14.9.3
         | cache-extension                          Paragraphe 14.9.6

cache-extension = jeton [ "=" ( jeton | quoted-string ) ]
```

Lorsqu'une directive apparaît sans un paramètre 1#field-name, la directive s'applique à toute la demande ou réponse. Lorsqu'une telle directive apparaît avec un paramètre 1#field-name, elle ne s'applique qu'au ou aux champs nommés, et non au reste de la demande ou réponse. Ce mécanisme accepte l'extensibilité ; les mises en œuvre des futures versions du protocole HTTP pourraient appliquer ces directives à des champs d'en-tête non définis dans HTTP/1.1.

Les directives de commande d'antémémoire peuvent être rangées dans les catégories générales suivantes :

- Restrictions sur ce qui est mettable en antémémoire ; celles-ci ne peuvent être imposées que par le serveur d'origine.
- Restrictions sur ce qui peut être mémorisé par une antémémoire ; celles-ci peuvent être imposées soit par le serveur d'origine, soit par l'agent d'utilisateur.
- Modifications du mécanisme d'expiration de base ; elles peuvent être imposées après le serveur d'origine ou l'agent d'utilisateur.
- Commandes sur la revalidation et le rechargement d'antémémoire ; elles ne peuvent être imposées que par l'agent d'utilisateur.
- Commandes sur la transformation des entités.
- Extensions au système de mise en antémémoire.

14.9.1 Ce qui est mettable en antémémoire

Par défaut, une réponse est mettable en antémémoire si les exigences de la méthode de demande, les champs d'en-tête de demande, et l'état de la réponse indiquent que c'est mettable en antémémoire. Le paragraphe 13.4 résume ces valeurs par défaut pour la possibilité de mettre en antémémoire. Les directives de réponse Cache-Control suivantes permettent à un serveur d'origine de subroger la la possibilité de mettre en antémémoire par défaut d'une réponse :

public
Indique que la réponse PEUT être mise en antémémoire par toute antémémoire, même si elle devrait normalement être non mettable en antémémoire ou seulement au sein d'une antémémoire non partagée. (Voir aussi Authorization, au paragraphe 14.8, pour des précisions.)

private
Indique que tout ou partie du message de réponse est destiné à un seul utilisateur et NE DOIT PAS être mémorisé par une antémémoire partagée. Ceci permet à un serveur d'origine d'établir que les parties spécifiées de la réponse sont destinées à un seul utilisateur et ne sont pas une réponse valide pour des demandes d'autres utilisateurs. Une antémémoire privée (non partagée) PEUT mémoriser la réponse.

Note : Cet usage du mot private ne contrôle que l'endroit où la réponse peut être mise en antémémoire, et ne peut pas assurer la confidentialité du contenu du message.

no-cache
Si la directive no-cache ne spécifie pas de nom de champ, une antémémoire NE DOIT PAS alors utiliser la réponse pour satisfaire une demande ultérieure sans revalidation auprès du serveur d'origine. Ceci permet à un serveur d'origine d'empêcher la mise en antémémoire même par des mémoires qui ont été configurées pour retourner des réponses périmées aux demandes des clients.

Si la directive no-cache spécifie un ou plusieurs noms de champ, une antémémoire PEUT alors utiliser la réponse pour satisfaire une demande ultérieure, sous réserve de toute autre restriction sur la mise en antémémoire. Cependant, le ou les noms de champ spécifiés NE DOIVENT PAS être envoyés dans la réponse à une demande ultérieure sans revalidation réussie auprès du serveur d'origine. Ceci permet à un serveur d'origine d'empêcher la réutilisation de certains champs d'en-tête dans une réponse, tout en permettant la mise en antémémoire du reste de la réponse.

Note : La plupart des antémémoires HTTP/1.0 ne reconnaîtront pas cette directive ou n'y obéiront pas.

14.9.2 Ce qui peut être mémorisé par les antémémoires

no-store
L'objet de la directive no-store est d'empêcher la libération ou rétention par inadvertance d'informations sensibles (par exemple, sur des bandes de sauvegarde). La directive no-store s'applique au message entier, et PEUT être envoyée dans une réponse ou dans une demande. Si elle est envoyée dans une demande, une antémémoire NE DOIT mémoriser aucune partie ni de cette demande ni de sa réponse. Si elle est envoyée dans une réponse, une antémémoire NE DOIT mémoriser aucune partie ni de cette réponse ni de la demande qui l'a provoquée. Cette directive s'applique aussi bien aux antémémoires partagées que non partagées. "NE DOIT PAS mémoriser" signifie dans ce contexte que l'antémémoire NE DOIT PAS intentionnellement mémoriser les informations dans un stockage non volatile, et DOIT faire de son mieux pour retirer les informations du stockage volatile aussi vite que possible après l'avoir transmise.

Même lorsque cette directive est associée à une réponse, les utilisateurs peuvent explicitement mémoriser une telle réponse en-dehors du système d'antémémoire (par exemple, avec un dialogue"Save As"). Les mémoires tampon d'historique PEUVENT mémoriser de telles réponses au titre de leur fonctionnement normal.

L'objet de cette directive est de satisfaire aux exigences établies de certains utilisateurs et auteurs de services qui se préoccupent de la délivrance accidentelle d'informations via des accès non prévus à des structures de données en antémémoire. Bien que l'utilisation de cette directive puisse améliorer dans certains cas la confidentialité, on doit souligner que ce n'est EN AUCUN CAS un mécanisme fiable ou suffisant pour assurer la confidentialité. En particulier, des antémémoires trompeuses ou compromises pourraient ne pas reconnaître ou obéir à cette directive, et les réseaux de communications pourraient être vulnérables à l'espionnage.

14.9.3 Modifications du mécanisme d'expiration de base

L'heure d'expiration d'une entité PEUT être spécifiée par le serveur d'origine en utilisant l'en-tête Expires (voir au paragraphe 14.21). Autrement, elle PEUT être spécifiée en utilisant la directive max-âge dans une réponse. Lorsque la directive de commande d'antémémoire max-âge est présente dans une réponse d'antémémoire, la réponse est périmée si l'âge actuel est supérieur à la valeur d'âge donnée (en secondes) au moment de la nouvelle demande pour cette ressource. La directive max-âge sur une réponse implique que la réponse soit

mettable en antémémoire (c'est-à-dire, "public") sauf si une autre directive d'antémémoire, plus restrictive, est aussi présente.

Si une réponse inclut à la fois un en-tête Expires et une directive max-âge, la directive max-âge subroge l'en-tête Expires, même si l'en-tête Expires est plus restrictif. Cette règle permet à un serveur d'origine de fournir, pour une réponse donnée, une durée d'expiration plus longue à une antémémoire HTTP/1.1 (ou plus récente) qu'une antémémoire HTTP/1.0. Ceci peut être utile si certaines antémémoires HTTP/1.0 calculent de façon impropre les âges ou les heures d'expiration, à cause peut-être d'horloges désynchronisées.

De nombreuses mises en œuvre d'antémémoire HTTP/1.0 vont traiter une valeur Expires inférieure ou égale à la valeur de la réponse Date comme équivalente à la directive de réponse Cache-Control "no-cache". Si une antémémoire HTTP/1.1 reçoit une telle réponse, et si la réponse n'inclut pas un champ d'en-tête Cache-Control, elle DEVRAIT considérer la réponse comme étant non mettable en antémémoire afin de conserver la compatibilité avec les serveurs HTTP/1.0.

Note : Un serveur d'origine peut souhaiter utiliser un dispositif de commande d'antémémoire HTTP relativement nouveau, comme la directive "private", sur un réseau incluant des antémémoires plus anciennes qui ne comprennent pas un tel dispositif. Le serveur d'origine devra combiner le nouveau dispositif avec un champ Expires dont la valeur est inférieure ou égale à la valeur de Date. Ceci empêchera les antémémoires plus anciennes de mettre à tort la réponse en mémoire.

s-maxage
Si une réponse inclut une directive s-maxage, pour une antémémoire partagée (mais non une mémoire privée), l'âge maximum spécifié par cette directive subroge l'âge maximum spécifié par la directive max-âge ou par l'en-tête Expires. La directive s-maxage implique aussi la sémantique de la directive proxy-revalidate (voir au paragraphe 14.9.4), c'est-à-dire que l'antémémoire partagée ne doit pas utiliser l'entrée après qu'elle soit devenue périmée pour répondre à une demande ultérieure sans l'avoir d'abord revalidée auprès du serveur d'origine. La directive s-maxage est toujours ignorée par une antémémoire privée.

Noter que les plus anciennes antémémoires, qui ne se conforment pas à la présente spécification, ne mettent en œuvre aucune directive de commande d'antémémoire. Un serveur d'origine qui souhaite utiliser une directive de commande d'antémémoire qui restreint, mais n'empêche pas, la mise en antémémoire par une antémémoire conforme à HTTP/1.1 PEUT exploiter l'exigence que la directive max-âge subroge l'en-tête Expires, et le fait que les antémémoires conformes à des versions antérieures à HTTP/1.1 n'observent pas la directive max-âge.

D'autres directives permettent à un agent d'utilisateur de modifier le mécanisme d'expiration de base. Ces directives PEUVENT être spécifiées sur une demande :

max-âge
Indique que le client est d'accord pour accepter une réponse dont l'âge n'est pas supérieur à celui spécifié en secondes. Sauf si la directive max-stale est aussi incluse, le client n'est pas d'accord pour accepter une réponse périmée.

min-fresh
Indique que le client est d'accord pour accepter une réponse dont la durée de vie de fraîcheur n'est pas inférieure à son âge actuel plus la durée spécifiée en secondes. C'est-à-dire que le client veut une réponse qui sera encore fraîche pour au moins le nombre de secondes spécifié.

max-stale
Indique que le client est d'accord pour accepter une réponse qui a dépassé son heure d'expiration. Si max-stale a reçu une valeur, le client est alors d'accord pour accepter une réponse qui a dépassé son heure d'expiration de moins que le nombre de secondes spécifié. Si aucune valeur n'est allouée à max-stale, le client est alors d'accord pour accepter une réponse périmée de n'importe quel âge.

Si une antémémoire retourne une réponse périmée, soit à cause d'une directive max-stale sur une demande, soit parce que l'antémémoire est configurée pour subroger la durée d'expiration d'une réponse, l'antémémoire DOIT attacher un en-tête Warning à la réponse périmée, en utilisant l'avertissement 110 (Réponse périmée).

Une antémémoire PEUT être configurée pour retourner des réponses périmées sans validation, mais seulement si cela n'entre pas en conflit avec une exigence de niveau "DOIT" concernant la validation d'antémémoire (par exemple, une directive de commande d'antémémoire "must-revalidate").

Si la nouvelle demande et l'entrée d'antémémoire incluent toutes deux les directives "max-âge", la plus faible des deux valeurs est utilisée pour déterminer la fraîcheur de l'entrée d'antémémoire pour cette demande.

14.9.4 Revalidation d'antémémoire et contrôles de rechargement

Un agent d'utilisateur peut parfois vouloir ou avoir besoin d'insister pour qu'une antémémoire revalide son entrée d'antémémoire auprès du serveur d'origine (et pas seulement avec l'antémémoire suivante sur le chemin du serveur d'origine) ou de recharger son entrée d'antémémoire à partir du serveur d'origine. La revalidation de bout en bout peut être nécessaire si l'antémémoire ou le serveur d'origine a surestimé le temps d'expiration de la réponse d'antémémoire. Le rechargement de bout en bout peut être nécessaire si l'entrée d'antémémoire a été corrompue pour une raison ou une autre.

La revalidation de bout en bout peut être demandée lorsque le client n'a pas sa propre copie locale mémorisée, auquel cas on l'appelle "revalidation de bout en bout non spécifiée", ou lorsque le client a mémorisé une copie locale, auquel cas on l'appelle "revalidation de bout en bout spécifique".

Le client peut spécifier ces trois sortes d'actions en utilisant les directives de demande Cache-Control :

End-to-end reload *(rechargement de bout en bout)*
La demande comporte une directive de contrôle d'antémémoire "no-cache" ou, pour la compatibilité avec les clients HTTP/1.0, "Pragma: no-cache". Les noms de champ NE DOIVENT PAS être inclus avec la directive no-cache dans une demande. Le serveur NE DOIT PAS utiliser une copie d'antémémoire lorsqu'il répond à une telle demande.

Specific end-to-end revalidation *(revalidation spécifique de bout en bout)*
La demande comporte une directive de commande d'antémémoire "max-âge=0", qui force chaque antémémoire le long du chemin vers le serveur d'origine à revalider sa propre entrée, s'il en est, auprès de la prochaine antémémoire ou du prochain serveur. La demande initiale inclut une condition de validation d'antémémoire avec le valideur actuel du client.

Unspecified end-to-end revalidation *(revalidation de bout en bout non spécifiée)*
La demande inclut une directive de commande d'antémémoire "max-âge=0", qui force chaque antémémoire le long du chemin vers le serveur d'origine à revalider sa propre entrée, s'il en est, auprès de la prochaine antémémoire ou du prochain serveur. La demande initiale n'inclut pas de condition de validation d'antémémoire ; la première antémémoire (s'il en est) le long du chemin qui contient une entrée d'antémémoire pour cette ressource inclut une condition de validation d'antémémoire avec son valideur actuel.

max-âge *(âge maximal)*
Lorsqu'une antémémoire intermédiaire est forcée, au moyen d'une directive max-âge=0, à revalider sa propre entrée d'antémémoire, et que le client a fourni son propre valideur dans la demande, le valideur fourni peut différer du valideur actuellement mémorisé avec l'entrée d'antémémoire. Dans ce cas, l'antémémoire PEUT utiliser l'un des valideurs en faisant sa propre demande sans affecter la transparence sémantique.

Cependant, le choix du valideur peut affecter les performances. La meilleure approche pour la mémoire intermédiaire est d'utiliser son propre valideur en faisant la demande. Si le serveur répond par 304 (Non modifiée), l'antémémoire peut alors retourner sa copie maintenant validée au client avec une réponse 200 (OK). Si le serveur répond par une nouvelle entité avec un valideur de cache, l'antémémoire intermédiaire peut cependant comparer le valideur retourné avec celui fourni dans la demande du client, en utilisant la fonction de comparaison forte. Si le valideur du client est égal à celui du serveur d'origine, l'antémémoire intermédiaire retourne simplement 304 (Non modifiée). Autrement, elle retourne la nouvelle entité avec une réponse 200 (OK).

Si une demande inclut la directive no-cache, elle NE DEVRAIT PAS inclure min-fresh, max-stale, ou max-âge.

only-if-cached *(seulement si en antémémoire)*
Dans certains cas, comme lors d'une très mauvaise connexité réseau, un client peut vouloir qu'une antémémoire ne retourne que les réponses qu'elle a actuellement en mémoire, et ne fasse pas de rechargement ou de revalidation auprès du serveur d'origine. Pour ce faire, le client peut inclure la directive only-if-cached dans une demande. Si elle reçoit cette directive, une antémémoire DEVRAIT répondre en utilisant une entrée d'antémémoire cohérente avec les autres contraintes de la demande, ou répondre par un

état 504 (Expiration de passerelle). Cependant, si un groupe d'antémémoires doit fonctionner comme un système unifié avec une bonne connexité interne, une telle demande PEUT être transmise au sein de ce groupe d'antémémoires.

must-revalidate *(doit revalider)*
Comme une antémémoire PEUT être configurée pour ignorer le temps d'expiration spécifié d'un serveur, et du fait qu'une demande de client PEUT inclure une directive max-stale (qui a un effet similaire), le protocole inclut aussi un mécanisme permettant au serveur d'origine de requérir la revalidation d'une entrée d'antémémoire sur toute utilisation ultérieure. Lorsque la directive must-revalidate est présente dans une réponse reçue par une antémémoire, celle-ci NE DOIT PAS utiliser l'entrée après sa péremption pour répondre à une demande ultérieure sans d'abord la revalider auprès du serveur d'origine. (C'est-à-dire que l'antémémoire DOIT faire une revalidation de bout en bout à chaque fois, si, sur la seule base de la valeur Expires ou max-âge du serveur d'origine, la réponse d'antémémoire est périmée.)

La directive must-revalidate est nécessaire pour prendre en charge un fonctionnement fiable pour certaines caractéristiques de protocole. Dans toutes les circonstances, une antémémoire HTTP/1.1 DOIT obéir à la directive must-revalidate ; en particulier, si l'antémémoire ne peut pas atteindre le serveur d'origine pour une raison quelconque, elle DOIT générer une réponse 504 (Expiration de passerelle).

Les serveurs DEVRAIENT envoyer la directive must-revalidate si et seulement si l'échec de revalidation d'une demande sur l'entité pourrait résulter en un fonctionnement incorrect, comme la non exécution sans notification d'une transaction financière. Les receveurs NE DOIVENT PAS entreprendre d'action automatique qui violerait cette directive, et NE DOIVENT PAS fournir automatiquement une copie non validée de l'entité si la revalidation échoue.

Bien que ce ne soit pas recommandé, les agents d'utilisateur qui fonctionnent sous des contraintes de connexité sévères PEUVENT violer cette directive mais, s'ils le font, DOIVENT explicitement avertir l'utilisateur qu'une réponse non validée a été fournie. L'avertissement DOIT être fourni sur chaque accès non validé, et DEVRAIT exiger une confirmation explicite de l'utilisateur.

proxy-revalidate *(revalidation du mandataire)*
La directive proxy-revalidate a la même signification que la directive must-revalidate, sauf qu'elle ne s'applique pas aux antémémoires d'agent d'utilisateur non partagées. Elle peut être utilisée sur une réponse à une demande authentifiée pour permettre à l'antémémoire de l'utilisateur de mémoriser la réponse et de la retourner ultérieurement sans avoir besoin de la revalider (car elle a déjà été authentifiée une fois par cet utilisateur), tout en exigeant des mandataires qui desservent beaucoup d'utilisateurs de se revalider à chaque fois (afin de s'assurer que chaque utilisateur a été authentifié). Noter que de telles réponses authentifiées ont aussi besoin de la directive de commande d'antémémoire publique afin de leur permettre d'être mises en mémoire.

14.9.5 Directive No-Transform

Les mises en œuvre de no-transform d'antémémoires intermédiaires (mandataires) ont estimé utile de convertir le type de support de certains corps d'entité. Un mandataire non transparent pourrait, par exemple, convertir des formats d'image afin d'économiser de l'espace d'antémémoire ou de réduire la quantité de trafic sur une liaison lente.

Des problèmes de fonctionnement sérieux surviennent cependant, lorsque ces transformations sont appliquées à des corps d'entité destinés à certains types d'applications. Par exemple, des applications d'imagerie médicale, d'analyse de données scientifiques et celles utilisant l'authentification de bout en bout, dépendent toutes de la réception d'un corps d'entité qui doit être identique bit par bit au corps d'entité d'origine.

Donc, si un message comporte la directive no-transform, une antémémoire intermédiaire ou mandataire NE DOIT PAS changer les en-têtes dont la liste figure au paragraphe 13.5.2 comme étant sujets à la directive no-transform. Ceci implique que l'antémémoire ou mandataire NE DOIT changer aucun aspect du corps d'entité spécifié par ces en-têtes, y compris la valeur du corps d'entité lui-même.

14.9.6 Extensions de commande d'antémémoire

Le champ d'en-tête Cache-Control peut être étendu par l'utilisation d'un ou plusieurs jetons d'extension d'antémémoire, chacun ayant une valeur allouée facultative. Les extensions informatives (celles qui n'exigent

pas de changement du comportement de l'antémémoire) PEUVENT être ajoutées sans changer la sémantique des autres directives. Les extensions comportementales sont conçues pour fonctionner en agissant comme modificateurs de la base existante des directives d'antémémoire. Les directives nouvelles et les directives standard sont toutes deux fournies, car les applications qui ne comprennent pas la nouvelle directive vont revenir par défaut au comportement spécifié par la directive standard, et celles qui comprennent la nouvelle directive vont la reconnaître comme modifiant les exigences associées à la directive standard. De cette façon, les extensions aux directives de commande de l'antémémoire peuvent être faites sans exiger de changement au protocole de base.

Ce mécanisme d'extension dépend du fait que les antémémoires HTTP obéissent toutes aux directives de commande d'antémémoire définies pour leur version HTTP d'origine, obéissant à certaines extensions, et ignorant toutes les directives qu'elles ne comprennent pas.

Par exemple, considérons une directive hypothétique de nouvelle réponse appelée community qui agit comme modificateur de la directive private. On définit cette nouvelle directive comme signifiant que, en plus de toute antémémoire non partagée, toute antémémoire qui n'est partagée que par des membres de la communauté nommée dans sa valeur peuvent mettre la réponse en antémémoire. Un serveur d'origine souhaitant permettre à la communauté UCI d'utiliser une autre réponse private dans leurs antémémoires partagées pourrait le faire en incluant

```
Cache-Control: private, community="UCI"
```

Une antémémoire voyant ce champ d'en-tête agira correctement même si l'antémémoire ne comprend pas l'extension d'antémémoire community, car elle verra aussi et comprendra la directive private et reviendra par défaut au comportement sûr.

Les directives d'antémémoire non reconnues DOIVENT être ignorées ; on suppose que vraisemblablement toute directive d'antémémoire non reconnue par une antémémoire HTTP/1.1 sera combinée à des directives standard (ou la possibilité de mettre en antémémoire par défaut de la réponse) de sorte que le comportement de l'antémémoire restera au minimum correct même si l'antémémoire ne comprend pas la ou les extensions.

14.10 Connection

Le champ d'en-tête général Connection permet à l'envoyeur de spécifier des options désirées pour cette connexion particulière et NE DOIT PAS être communiqué par les mandataires sur des connexions ultérieures.

L'en-tête Connection a la grammaire suivante :

```
Connection = "Connection" ":" 1#(connection-token)
connection-token  = jeton
```

Les mandataires HTTP/1.1 DOIVENT analyser le champ d'en-tête Connection avant l'envoi d'un message et, pour chaque jeton de connexion dans ce champ, retirer tout champ d'en-tête du message avec le même nom que le jeton de connexion. Les options de connexion sont signalées par la présence d'un jeton de connexion dans le champ d'en-tête Connection, et non par un ou des champs d'en-tête correspondants supplémentaires, car le champ d'en-tête supplémentaire peut n'être pas envoyé si il n'y a pas de paramètre associé à cette option de connexion.

Les en-têtes de message dont la liste figure dans l'en-tête Connection NE DOIVENT PAS inclure d'en-têtes de bout en bout, tels que Cache-Control.

HTTP/1.1 définit l'option de connexion "close" pour que l'envoyeur signale que la connexion sera close après achèvement de la réponse. Par exemple,

```
Connection: close
```

aussi bien dans les champs d'en-tête de demande ou de réponse indique que la connexion NE DEVRAIT PAS être considérée comme "persistante" (paragraphe 8.1) après l'achèvement de la demande/réponse en cours.

Les applications HTTP/1.1 qui ne prennent pas en charge les connexions persistantes DOIVENT inclure l'option de connexion "close" dans chaque message.

Un système qui reçoit un message HTTP/1.0 (ou version plus ancienne) qui inclut un en-tête Connection DOIT, pour chaque jeton de connexion dans ce champ, retirer et ignorer tout champ d'en-tête du message avec le même nom que le jeton de connexion. Ceci protège contre les erreurs de transmission de tels champs d'en-tête par des mandataires pré HTTP/1.1. Voir au paragraphe 19.6.2.

14.11 Content-Encoding

Le champ d'en-tête d'entité Content-Encoding *(codage du contenu)* dest utilisé comme modificateur du type de support. Lorsqu'il est présent, sa valeur indique que des codages de contenu supplémentaires ont été appliqués au corps d'entité, et donc, quels mécanismes de décodage doivent être appliqués afin d'obtenir le type de support référencé par le champ d'en-tête Content-Type. Content-Encoding est principalement utilisé pour permettre qu'un document soit compressé sans perdre l'identité de son type de support sous-jacent.

```
Content-Encoding  = "Content-Encoding" ":" 1#content-coding
```

Les codages de contenu sont définis au paragraphe 3.5. Un exemple de son utilisation est :

```
Content-Encoding: gzip
```

Le codage de contenu est une caractéristique de l'entité identifiée par l'URI de demande. Normalement, le corps d'entité est mémorisé avec ce codage et n'est décodé qu'avant la restitution ou utilisation analogue. Cependant, un mandataire non transparent PEUT modifier le codage de contenu si le nouveau codage est connu pour être acceptable par le receveur, sauf si la directive de commande d'antémémoire "no-transform" est présente dans le message.

Si le codage de contenu d'une entité n'est pas "identity", la réponse DOIT inclure un en-tête d'entité Content-Encoding (paragraphe 14.11) qui fait la liste des codages de contenu non-identity utilisés.

Si le codage de contenu d'une entité dans un message de demande n'est pas acceptable par le serveur d'origine, le serveur DEVRAIT répondre par un code d'état de 415 (Type de support non accepté).

Si plusieurs codages ont été appliqués à une entité, la liste des codages de contenu DOIT figurer dans l'ordre dans lequel ils ont été appliqués. Des informations supplémentaires sur les paramètres de codage PEUVENT être fournies par d'autres champs d'en-tête d'entité non définis par la présente spécification.

14.12 Content-Language

Le champ d'en-tête d'entité Content-Language décrit le ou les langages naturels des destinataires prévus pour l'entité incluse. Noter que cela peut n'être pas équivalent à tous les langages utilisés au sein du corps d'entité.

```
Content-Language  = "Content-Language" ":" 1#language-tag
```

Les étiquettes de langues sont définies au paragraphe 3.10. Le principal objet de Content-Language est de permettre à l'utilisateur d'identifier et différencier les entités conformément à la propre langue de préférence de l'utilisateur. Et donc, si le contenu du corps n'est destiné qu'à une audience parlant danois, le champ approprié est :

```
Content-Language: da
```

Si aucun Content-Language n'est spécifié, la valeur par défaut est que le contenu est destiné à toutes les audiences de langues. Cela peut signifier que l'envoyeur ne le considère pas comme étant spécifique d'aucun langage naturel, ou que l'envoyeur ne sait pas à quel langage il est destiné.

Plusieurs langages PEUVENT figurer sur la liste des contenus qui sont destinés à plusieurs audiences. Par exemple, une restitution du "Traité de Waitangi," présenté simultanément dans la version Maori originale et en version anglaise, nécessiterait :

```
Content-Language: mi, en
```

Cependant, le fait que plusieurs langues soient présentes au sein d'une entité ne signifie pas qu'elle soit destinée à plusieurs audiences linguistiques. Un exemple serait celui d'un manuel de débutant en langue, tel que "Première leçon de latin," qui est clairement destiné à être utilisé par une audience francophone. Dans ce cas, le Content-Language ne contiendrait à juste titre que "fr".

Content-Language PEUT être appliqué à tout type de support -- il n'est pas limité aux documents textuels.

14.13 Content-Length

Le champ d'en-tête d'entité Content-Length *(longueur du contenu)* indique la taille du corps d'entité, en nombre décimal d'octets, envoyé au receveur ou, dans le cas de la méthode HEAD, la taille du corps d'entité qui aurait été envoyé si la demande avait été GET.

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

Un exemple est :

```
Content-Length: 3495
```

Les applications DEVRAIENT utiliser ce champ pour indiquer la longueur de transfert du corps de message, sauf si c'est interdit par les règles du paragraphe 4.4.

Tout Content-Length supérieur ou égal à zéro est une valeur valide. Le paragraphe 4.4 décrit comment déterminer la longueur d'un corps de message si Content-Length n'est pas donné.

Noter que la signification de ce champ est notablement différente de la définition correspondante dans MIME, où c'est un champ facultatif utilisé dans le type de contenu "message/external-body". Dans HTTP, il DEVRAIT être envoyé chaque fois que la longueur du message peut être déterminée avant d'être transféré, sauf si c'est interdit par les règles du paragraphe 4.4.

14.14 Content-Location

Le champ d'en-tête d'entité Content-Location *(localisation du contenu)* PEUT être utilisé pour fournir la localisation de ressource pour l'entité incluse dans le message lorsque cette entité est accessible à partir d'une localisation distincte de l'URI de la ressource demandée. Un serveur DEVRAIT fournir un Content-Location pour la variante correspondant à l'entité de réponse ; en particulier dans le cas où une ressource a plusieurs entités associées, et où ces entités ont en fait des localisations séparées par lesquelles on peut y avoir individuellement accès, le serveur DEVRAIT fournir un Content-Location pour la variante particulière retournée.

```
Content-Location = "Content-Location" ":" ( absoluteURI | relativeURI )
```

La valeur de Content-Location définit aussi l'URI de base pour l'entité.

La valeur Content-Location ne remplace pas l'URI de la demande d'origine ; elle est une simple déclaration de la localisation de la ressource correspondant à cette entité particulière au moment de la demande. Des demandes futures PEUVENT spécifier l'URI de Content-Location comme URI de demande si on désire identifier la source de cette entité particulière.

Une antémémoire ne peut pas supposer qu'une entité qui a un Content-Location différent de l'URI utilisé pour la restituer peut être utilisée pour répondre ultérieurement à des demandes sur cet URI de Content-Location. Cependant, le Content-Location peut être utilisé pour différencier plusieurs entités restituées à partir d'une seule ressource demandée, comme décrit au paragraphe 13.6.

Si le Content-Location se rapporte à un URI, l'URI relatif est interprété par rapport à l'URI de demande.

La signification de l'en-tête Content-Location dans des demandes PUT ou POST est indéfini ; les serveurs ont la liberté de l'ignorer dans ces cas là.

14.15 Content-MD5

Le champ d'en-tête d'entité Content-MD5, comme défini à la RFC 1864 [23], est un résumé MD5 du corps d'entité pour les besoins de fourniture d'une vérification d'intégrité de message de bout en bout (MIC, *message integrity check*) du corps d'entité. (Note : un MIC est bon pour la détection de modifications accidentelles sur le corps d'entité en transit, mais n'est pas éprouvé contre des attaques mal intentionnées.)

```
Content-MD5   = "Content-MD5" ":" md5-digest
md5-digest   = <résumé MD5 de 128 bits en base64 conformément à la RFC 1864>
```

Le champ d'en-tête Content-MD5 PEUT être généré par un serveur d'origine ou client pour fonctionner comme une vérification d'intégrité du corps d'entité. Seuls les serveurs d'origine ou clients PEUVENT générer le champ d'en-tête Content-MD5 ; les mandataires et passerelles NE DOIVENT PAS le générer, car cela nuirait à sa valeur comme vérification d'intégrité de bout en bout. Tout receveur du corps d'entité, y compris les passerelles et mandataires, PEUT vérifier que la valeur du résumé dans ce champ d'en-tête correspond à celle qu'a reçue le corps d'entité.

Le résumé MD5 est calculé sur la base du contenu du corps d'entité, incluant tout codage de contenu qui lui a été appliqué, mais non inclus tout codage de transfert appliqué au corps de message. Si le message est reçu avec un codage de transfert, ce codage DOIT être retiré avant de vérifier la valeur de Content-MD5 par rapport à celle de l'entité reçue.

Il en résulte que le résumé est calculé sur les octets du corps d'entité exactement comme, et dans l'ordre où, il aurait été envoyé si aucun codage de transfert n'avait été appliqué.

HTTP étend la RFC 1864 pour permettre que le résumé soit calculé pour les types de support composites de MIME (par exemple, multipart/* et message/rfc822), mais cela ne change pas la façon de calculer le résumé comme défini au paragraphe précédent.

Il y a plusieurs conséquences à cela. Le corps d'entité pour les types composites PEUT contenir de nombreuses parties de corps, chacune avec ses propres en-têtes MIME et HTTP (y compris les en-têtes Content-MD5, Content-Transfer-Encoding, et Content-Encoding). Si une partie de corps a un en-tête Content-Transfer-Encoding ou Content-Encoding, on suppose que le codage est appliqué au contenu de la partie de corps, et que la partie de corps est incluse dans le résumé Content-MD5 comme il est -- c'est-à-dire, après l'application. Le champ d'en-tête Transfer-Encoding n'est pas admis dans les parties de corps.

La conversion de toutes les coupures de ligne en CRLF NE DOIT PAS être faite avant de calculer ou vérifier le résumé : la convention de rupture de ligne utilisée dans le texte réellement transmis DOIT être laissée inaltérée lors du calcul du résumé.

Note Alors que la définition de Content-MD5 est exactement la même pour HTTP que dans la RFC 1864 pour les corps d'entité MIME, il y a plusieurs différences d'application entre le Content-MD5 des corps d'entité HTTP et MIME. Une de ces différences est que HTTP, à la différence de MIME, n'utilise pas Content-Transfer-Encoding, et utilise Transfer-Encoding et Content-Encoding. Une autre différence est que HTTP utilise plus fréquemment les types de contenus binaires que MIME, et il faut noter que dans de tels cas, l'ordre binaire utilisé pour calculer le résumé est l'ordre de transmission d'octet défini pour le type. Enfin, HTTP permet la transmission de types de textes avec toutes les différentes conventions de rupture de ligne et non seulement la forme canonique utilisant CRLF.

14.16 Content-Range

L'en-tête d'entité Content-Range *(gamme de contenu)* est envoyé avec un corps d'entité partiel pour spécifier où le corps partiel devrait être appliqué dans le corps d'entité complet. Les unités de gamme sont définies au paragraphe 3.12.

```
Content-Range = "Content-Range" ":" content-range-spec

content-range-spec      = byte-content-range-spec
byte-content-range-spec = bytes-unit SP byte-range-resp-spec "/" ( instance-length | "*" )
byte-range-resp-spec    = (first-byte-pos "-" last-byte-pos) | "*"
instance-length         = 1*DIGIT
```

L'en-tête DEVRAIT indiquer la longueur totale du corps d'entité complet, sauf si cette longueur est inconnue ou difficile à déterminer. Le caractère astérisque "*" signifie que la longueur de l'instance est inconnue au moment où la réponse est générée.

À la différence des valeurs de spécificateurs de gamme d'octet (voir au paragraphe 14.35.1) un byte-range-resp-spec *(spécification de la gamme d'octets de la réponse)* DOIT seulement spécifier une gamme, et DOIT contenir les positions d'octet absolues à la fois pour le premier et le dernier octet de la gamme.

Un byte-content-range-spec *(spécification de la gamme des octets du contenu)* avec un byte-range-resp-spec dont la dernière valeur de last-byte-pos *(position du dernier octet)* est inférieure à sa valeur de first-byte-pos *(position du premier octet)* ou dont la valeur de instance-length *(longueur d'instance)* est inférieure ou égale à sa valeur de last-byte-pos, est invalide. Le receveur d'une byte-content-range-spec invalide DOIT l'ignorer ainsi que tout contenu transféré avec elle.

Un serveur qui envoie une réponse avec le code d'état 416 (Gamme demandée non satisfaisable) DEVRAIT inclure un champ Content-Range avec une byte-range-resp-spec de "*". La longueur d'instance spécifie la longueur actuelle de la ressource choisie. Une réponse avec le code d'état 206 (Contenu partiel) NE DOIT PAS inclure de champ Content-Range avec une byte-range-resp-spec de "*".

Exemples de valeurs de byte-content-range-spec, en supposant que l'entité contient un total de 1234 octets :

```
Les premiers 500 octets : octets 0-499/1234
Les seconds 500 octets : octets 500-999/1234
Tous les autres excepté les premiers 500 octets : octets 500-1233/1234
Les derniers 500 octets : octets 734-1233/1234
```

Lorsque un message HTTP comporte le contenu d'une seule gamme (par exemple, une réponse à une demande d'une seule gamme, ou à une demande d'un ensemble de gammes qui se chevauchent sans aucun trou) ce contenu est transmis avec un en-tête Content-Range, et a un en-tête Content-Length montrant le nombre d'octets réellement transférés. Par exemple,

```
HTTP/1.1 206 Partial content
Date: Wed, 15 Nov 1995 06:25:24 GMT
Last-Modified: Wed, 15 Nov 1995 04:58:08 GMT
Content-Range: bytes 21010-47021/47022
Content-Length: 26012
Content-Type: image/gif
```

Lorsque un message HTTP inclut le contenu de plusieurs gammes (par exemple, une réponse à une demande de plusieurs gammes sans recouvrement) elles sont transmises comme un message en plusieurs parties. Le type de support multiparties utilisé à cette fin est "multipart/byteranges" comme défini à l'appendice 19.2. Voir à l'appendice 19.6.3 le problème de la compatibilité.

Une réponse à une demande d'une seule gamme NE DOIT PAS être envoyée en utilisant le type de support multipart/byteranges. Une réponse à une demande de plusieurs gammes, dont le résultat est une seule gamme, PEUT être envoyée comme un type de support multipart/byteranges en une seule partie. Un client qui ne peut pas décoder un message multipart/byteranges NE DOIT PAS demander plusieurs gammes d'octet dans une seule demande.

Lorsqu'un client demande plusieurs gammes d'octet dans une seule demande, le serveur DEVRAIT les retourner dans l'ordre où elles apparaissent dans la demande.

Si le serveur ignore une byte-range-spec parce qu'elle est syntaxiquement invalide, le serveur DEVRAIT traiter la demande comme si le champ d'en-tête Range invalide n'existait pas. (Normalement, cela signifie de retourner une réponse 200 contenant l'entité complète).

Si le serveur reçoit une demande (autre qu'une comportant un champ d'en-tête de demande If-Range) avec un champ d'en-tête de demande Range non satisfaisable (c'est-à-dire que toutes les valeurs de byte-range-spec ont une valeur first-byte-pos supérieure à la longueur actuelle de la ressource choisie) il DEVRAIT retourner un code de réponse de 416 (Gamme demandée non satisfaisable) (paragraphe 10.4.17).

Note : Les clients ne peuvent pas dépendre des serveurs pour envoyer une réponse 416 (Gamme demandée

non satisfaisable) au lieu d'une réponse 200 (OK) pour un en-tête de demande Range non satisfaisable, car tous les serveurs ne mettent pas en œuvre cet en-tête de demande.

14.17 Content-Type

Le champ d'en-tête d'entité Content-Type indique le type de support que le corps d'entité a envoyé au receveur ou, dans le cas de la méthode HEAD, le type de support qui aurait été envoyé si la demande avait été GET.

```
Content-Type   = "Content-Type" ":" media-type
```

Les types de support sont définis au paragraphe 3.7. Un exemple du champ est :

```
Content-Type: text/html; charset=ISO-8859-4
```

Une discussion plus approfondie des méthodes d'identification du type de supports d'une entité est fournie au paragraphe 7.2.1.

14.18 Date

Le champ d'en-tête général Date représente la date et l'heure à laquelle le message a eu son origine. Il a la même sémantique que orig-date dans la RFC 822. La valeur du champ est une date HTTP, comme décrit au paragraphe 3.3.1 ; il DOIT être envoyé dans le format de date de la RFC 1123 [8].

```
Date  = "Date" ":" HTTP-date
```

Exemple :

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

Les serveurs d'origine DOIVENT inclure un champ d'en-tête Date dans toutes les réponses, sauf dans les cas suivants :

1. Si le code d'état de réponse est 100 (Continue) ou 101 (Changement de protocoles), la réponse PEUT inclure un champ d'en-tête Date, au choix du serveur.

2. Si le code d'état de réponse comporte une erreur du serveur, par exemple 500 (Erreur interne du serveur) ou 503 (Service indisponible), et qu'il est difficile ou impossible de générer une Date valide.

3. Si le serveur n'a pas d'horloge qui puisse fournir une approximation raisonnable de l'heure actuelle, sa réponse NE DOIT PAS inclure de champ d'en-tête Date. Dans ce cas, on DOIT suivre les règles du paragraphe 14.18.1.

Un message reçu sans champ d'en-tête Date DOIT en recevoir une par le receveur si le message doit être mis en antémémoire par ce receveur ou passé par une passerelle via un protocole qui exige une Date. Une mise en œuvre HTTP sans horloge NE DOIT PAS mettre de réponse en antémémoire sans la revalider à chaque utilisation. Une antémémoire HTTP, et en particulier une antémémoire partagée, DEVRAIT utiliser un mécanisme, tel que NTP [28], pour synchroniser son horloge avec une norme externe fiable.

Les clients DEVRAIENT seulement envoyer un champ d'en-tête Date dans les messages qui comportent un corps d'entité, comme dans le cas des demandes PUT et POST, et même là, c'est facultatif. Un client sans horloge NE DOIT PAS envoyer de champ d'en-tête Date dans une demande.

Le HTTP-date envoyé dans un en-tête Date NE DEVRAIT PAS représenter une date et heure ultérieure à la génération du message. Il DEVRAIT représenter la meilleure approximation disponible de la date et heure de la génération du message, sauf si la mise en œuvre n'a pas les moyens de générer une date et heure raisonnablement précise. En théorie, la date devrait représenter le moment précédent immédiatement celui où l'entité est générée. En pratique, la date peut être générée à tout moment durant la génération du message sans affecter sa valeur sémantique.

14.18.1 Fonctionnement de serveur d'origine sans horloge

Certaines mises en œuvre de serveur d'origine peuvent n'avoir pas d'horloge disponible. Un serveur d'origine sans horloge NE DOIT PAS allouer de valeurs Expires ou Last-Modified à une réponse, sauf si ces valeurs sont associées à la ressource par un système ou utilisateur ayant une horloge fiable. Elles PEUVENT allouer une valeur Expires connue, au moment ou avant l'heure de configuration du serveur, pour être dans le passé (ceci permet la "pré-expiration" des réponses sans mémoriser de valeurs Expires distinctes pour chaque ressource).

14.19 ETag

Le champ d'en-tête de réponse ETag fournit la valeur actuelle de l'étiquette d'entité pour la variante demandée. Les en-têtes utilisés avec les étiquettes d'entité sont décrits aux paragraphes 14.24, 14.26 et 14.44. L'étiquette d'entité PEUT être utilisée pour comparaison avec d'autres entités provenant de la même ressource (voir au paragraphe 13.3.3).

```
ETag = "ETag" ":" étiquette d'entité
```

Exemples :

```
ETag: "xyzzy"
ETag: W/"xyzzy"
ETag: ""
```

14.20 Expect

Le champ d'en-tête de demande Expect est utilisé pour indiquer que des comportements de serveur particuliers sont exigés par le client.

```
Expect  = "Expect" ":" 1#expectation
expectation  = "100-continue" | expectation-extension
expectation-extension  = jeton [ "=" ( jeton | quoted-string )  *expect-params ]
expect-params  = ";" jeton [ "=" ( jeton | quoted-string ) ]
```

Un serveur qui ne comprend pas, ou est incapable de se conformer à une des valeurs attendues dans le champ Expect d'une demande, DOIT répondre par le code d'erreur approprié. Le serveur DOIT répondre par un état 417 (Échec de l'attente) si une des attentes ne peut être satisfaite, et si il y a d'autres problèmes avec la demande, d'autres états 4xx.

Ce champ d'en-tête est défini avec une syntaxe extensible pour permettre des extensions à l'avenir. Si un serveur reçoit une demande contenant un champ Expect qui inclut une expectation-extension qu'il ne prend pas en charge, il DOIT répondre par un état 417 (Échec de l'attente).

La comparaison des valeurs d'attente est insensible à la casse pour les jetons qui ne sont pas entre guillemets (y compris le jeton 100 (Continue)) et est sensible à la casse pour les chaînes d'extensions d'attente entre guillemets.

Le mécanisme Expect est bond par bond : c'est-à-dire qu'un mandataire HTTP/1.1 DOIT retourner un état 417 (Échec de l'attente) si il a reçu une demande avec une attente qu'il ne peut satisfaire. Cependant, l'en-tête de demande Expect lui-même est de bout en bout ; il DOIT être transmis si la demande est transmise.

De nombreuses applications HTTP/1.0 et HTTP/1.1 plus anciennes ne comprennent pas l'en-tête Expect.

14.21 Expires

Le champ d'en-tête d'entité Expires donne la date/heure après laquelle la réponse est considérée comme périmée. Une entrée d'antémémoire périmée ne peut normalement pas être retournée par une antémémoire (mandataire d'antémémoire ou agent d'utilisateur d'antémémoire) sauf si elle s'est d'abord validée auprès du serveur d'origine (ou avec une antémémoire intermédiaire qui a une copie fraîche de l'entité). Voir au paragraphe 13.2 des précisions sur le modèle d'expiration.

La présence d'un champ Expires n'implique pas que la ressource d'origine va changer ou cesser d'exister à ce moment, ou avant ou après.

Le format est une date et heure absolue comme défini par HTTP-date au paragraphe 3.3.1 ; il DOIT être dans le format de date de la RFC 1123 :

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

Un exemple de son utilisation est :

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

Note Si une réponse inclut un champ Cache-Control avec la directive max-âge (voir au paragraphe 14.9.3), cette directive subroge le champ Expires.

Les clients et antémémoires HTTP/1.1 DOIVENT traiter les autres formats de date invalides, particulièrement ceux qui comportent la valeur "0", comme étant passés (c'est-à-dire, "déjà expirés").

Pour marquer une réponse comme "déjà expirée," un serveur d'origine envoie une date Expires égale à la valeur de l'en-tête Date. (Voir les règles de calcul d'expiration au paragraphe 13.2.4.)

Pour marquer une réponse comme "jamais expirée," un serveur d'origine envoie une date Expires d'approximativement un an à partir du moment de l'envoi de la réponse. Les serveurs HTTP/1.1 NE DEVRAIENT PAS envoyer de dates Expires de plus d'un an dans le futur.

La présence d'un champ d'en-tête Expires avec une valeur de date un peu postérieure à une réponse qui serait autrement non mettable en antémémoire par défaut indique que la réponse est mettable en antémémoire, sauf indication contraire par un champ d'en-tête Cache-Control (paragraphe 14.9).

14.22 From

Le champ d'en-tête de demande From, s'il est donné, DEVRAIT contenir une adresse de messagerie Internet pour l'utilisateur humain qui contrôle l'agent d'utilisateur demandeur. L'adresse DEVRAIT être utilisable par la machine, comme défini par "mailbox" dans la RFC 822 [9] mise à jour par la RFC 1123 [8] :

```
From   = "From" ":" mailbox
```

Un exemple en est :

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

Ce champ d'en-tête PEUT être utilisé pour les besoins de localisation et comme moyen d'identifier la source de demandes invalides ou non désirées. Il NE DEVRAIT PAS être utilisé comme une forme non sécurisée de protection d'accès. L'interprétation de ce champ est que la demande est à effectuer au nom de la personne désignée, qui accepte la responsabilité de la méthode effectuée. En particulier, les agents robots DEVRAIENT inclure cet en-tête de sorte que la personne responsable du fonctionnement du robot puisse être contactée si des problèmes surviennent à l'extrémité de réception.

L'adresse de messagerie Internet dans ce champ PEUT être séparée de l'hôte Internet qui a produit la demande. Par exemple, lorsque une demande est passée par un mandataire, l'adresse du producteur original DEVRAIT être utilisée.

Le client NE DEVRAIT PAS envoyer le champ d'en-tête From sans l'approbation de l'utilisateur, car cela pourrait entrer en conflit avec les intérêts de l'utilisateur en matière de confidentialité, ou avec la politique de sécurité du site. Il est fortement recommandé que l'utilisateur soit capable de désactiver, activer, et modifier la valeur de ce champ à tout moment avant une demande.

14.23 Host

Le champ d'en-tête de demande Host spécifie l'hôte Internet et le numéro d'accès *(port)* de la ressource

demandée, tels qu'obtenus de l'URI d'origine donné par l'utilisateur ou la ressource de référence (généralement un URL HTTP, comme décrit au paragraphe 3.2.2). La valeur du champ Host DOIT représenter l'autorité de nommage du serveur ou passerelle d'origine donnée par l'URL d'origine. Cela permet au serveur ou passerelle d'origine de faire la différence entre des URL qui ont des ambiguïtés internes, telles que la racine "/" d'un serveur pour plusieurs noms d'hôte sur une seule adresse IP.

```
Host = "Host" ":" host [ ":" port ] ; Paragraphe 3.2.2
```

Un "host" sans information d'accès en queue implique l'accès par défaut pour le service demandé (par exemple, "80" pour un URL HTTP). Par exemple, une demande sur le serveur d'origine pour <http://www.w3.org/pub/WWW/> inclurait à juste titre :

```
GET /pub/WWW/ HTTP/1.1
Host: www.w3.org
```

Un client DOIT inclure un champ d'en-tête Host dans tous les messages de demande HTTP/1.1. Si l'URI demandé n'inclut pas un nom d'hôte Internet pour le service demandé, le champ d'en-tête Host DOIT alors être donné avec une valeur vide. Un mandataire HTTP/1.1 DOIT s'assurer que tout message de demande qu'il transmet contient bien un champ d'en-tête Host approprié qui identifie le service demandé par le mandataire. Tous les serveurs HTTP/1.1 fondés sur Internet DOIVENT répondre par le code d'état 400 (Mauvaise demande) à tout message de demande HTTP/1.1 qui n'a pas de champ d'en-tête Host.

Voir aux paragraphes 5.2 et 19.6.1.1 les autres exigences relatives à Host.

14.24 If-Match

Le champ d'en-tête de demande If-Match (Si correspondance) est utilisé avec une méthode pour le rendre conditionnel. Un client qui a une ou plusieurs entités obtenues précédemment de la ressource peut vérifier qu'une de ces entités est en cours en incluant une liste de leurs étiquettes d'entité associées dans le champ d'en-tête If-Match. Les étiquettes d'entité sont définies au paragraphe 3.11. L'objet de ce dispositif est de permettre des mises à jour efficaces des informations en antémémoire avec une quantité minimum de redondance de transaction. Il est aussi utilisé, à la mise à jour des demandes, pour empêcher la modification par inadvertance de la mauvaise version d'une ressource. À titre de cas particulier, la valeur "*" correspond à toute entité en cours de la ressource.

```
If-Match = "If-Match" ":" ( "*" | 1#étiquette d'entité )
```

Si une étiquette d'entité quelconque correspond à l'étiquette d'entité de l'entité qui aurait été retournée dans la réponse à une demande GET similaire (sans l'en-tête If-Match) sur cette ressource, ou si "*" est donné et qu'une entité actuelle existe pour cette ressource, le serveur PEUT alors effectuer la méthode demandée comme si le champ d'en-tête If-Match n'existait pas.

Un serveur DOIT utiliser la fonction de comparaison forte (voir au paragraphe 13.3.3) pour comparer l'étiquette d'entités dans If-Match.

Si aucune des étiquettes d'entité ne correspond, ou si "*" est donné et qu'aucune entité en cours n'existe, le serveur NE DOIT PAS effectuer la méthode demandée, et DOIT retourner une réponse 412 (Échec de précondition). Ce comportement est très utile lorsque le client veut empêcher une méthode de mise à jour, telle que PUT, de modifier une ressource qui a changé depuis la dernière visite du client.

Si, sans le champ d'en-tête If-Match, la demande aurait eu pour résultat autre chose qu'un état 2xx ou 412, l'en-tête If-Match DOIT alors être ignoré.

La signification de "If-Match: *" est que la méthode DEVRAIT être effectuée si la représentation choisie par le serveur d'origine (ou par une antémémoire, éventuellement en utilisant le mécanisme Vary, voir au paragraphe 14.44) existe, et NE DOIT PAS être effectuée si la représentation n'existe pas.

Une demande destinée à mettre à jour une ressource (par exemple, un PUT) PEUT inclure un champ d'en-tête If-Match pour signaler que la méthode de demande NE DOIT PAS être appliquée si l'entité correspondant à la valeur de If-Match (une seule étiquette d'entité) n'est plus une représentation de cette ressource. Cela permet à l'utilisateur d'indiquer qu'il ne souhaite pas que la demande réussisse si la ressource a été changée à son insu.

Exemples :

```
If-Match: "xyzzy"
If-Match: "xyzzy", "r2d2xxxx", "c3piozzzz"
If-Match: *
```

Le résultat d'une demande qui aurait à la fois un champ d'en-tête If-Match et un champ d'en-tête If-None-Match ou If-Modified-Since n'est pas défini dans la présente spécification.

14.25 If-Modified-Since

Le champ d'en-tête de demande If-Modified-Since *(Si modifée depuis le)* est utilisé avec une méthode pour le rendre conditionnel : si la variante demandée n'a pas été modifiée depuis le moment spécifié dans ce champ, une entité ne sera pas retournée du serveur ; à la place, une réponse 304 (non modifiée) sera retournée sans corps de message.

```
If-Modified-Since = "If-Modified-Since" ":" HTTP-date
```

Un exemple du champ est :

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

Une méthode GET avec un en-tête If-Modified-Since et pas d'en-tête Range demande que l'entité identifiée ne soit transférée que si elle a été modifiée depuis la date donnée par l'en-tête If-Modified-Since. L'algorithme pour déterminer ceci comporte les cas suivants :

a) Si la demande résulterait normalement en autre chose qu'un état 200 (OK), ou si la date If-Modified-Since passée est invalide, la réponse est exactement la même que pour un GET normal. Une date postérieure à l'heure en cours du serveur est invalide.

b) Si la variante a été modifiée depuis la date de If-Modified-Since, la réponse est exactement la même que pour un GET normal.

c) Si la variante n'a pas été modifiée depuis la date valide de If-Modified-Since, le serveur DEVRAIT retourner une réponse 304 (Non modifiée).

L'objet de ce dispositif est de permettre des mises à jour efficaces d'informations conservées en antémémoire avec un minimum de redondance de transaction.

Note Le champ d'en-tête de demande Range modifie la signification de If-Modified-Since ; voir les précisions au paragraphe 14.35.

Note Les heures de If-Modified-Since sont interprétées par le serveur, dont l'horloge pourrait n'être pas synchronisée avec le client.

Note Lors du traitement d'un champ d'en-tête If-Modified-Since, certains serveurs vont utiliser une fonction de comparaison de date exacte, plutôt qu'une fonction "inférieur à", pour décider d'envoyer ou non une réponse 304 (Non modifiée). Pour obtenir les meilleurs résultats lors de l'envoi d'un champ d'en-tête If-Modified-Since pour une validation d'antémémoire, il est conseillé aux clients d'utiliser la chaîne de date exacte reçue dans un champ d'en-tête Last-Modified chaque fois que possible.

Note Si un client utilise une date arbitraire dans l'en-tête If-Modified-Since au lieu d'une date tirée du dernier en-tête Last-Modified pour la même demande, le client devrait être averti du fait que cette date sera interprétée par la compréhension de l'heure par le serveur. Le client devrait prendre en considération les horloges non synchronisées et les problèmes d'arrondi dus aux codages différents de l'heure entre le client et le serveur. Cela inclut la possibilité de conditions de concurrence si le document a changé entre le moment où il a été demandé pour la première fois et la date de If-Modified-Since d'une demande ultérieure, et la possibilité de problèmes liés au biais d'horloge si la date de If-Modified-Since est déduite de l'horloge du client sans correction par l'horloge du serveur. Les corrections pour les bases horaires différentes entre client et serveur sont au mieux approximées à cause de la latence du réseau.

Le résultat d'une demande qui a à la fois un champ d'en-tête If-Modified-Since et un champ d'en-tête If-Match ou If-Unmodified-Since n'est pas défini dans la présente spécification.

14.26 If-None-Match

Le champ d'en-tête de demande If-None-Match *(si aucune ne correspond)* est utilisé avec une méthode pour le rendre conditionnel. Un client qui a une ou plusieurs entités obtenues précédemment de la ressource peut vérifier qu'aucune de ces entités n'est en cours en incluant une liste de leurs étiquettes d'entité associées dans le champ d'en-tête If-None-Match. L'objet de ce dispositif est de permettre une mise à jour efficace des informations détenues en antémémoire avec un minimum de redondance de transaction. Il est aussi utilisé pour empêcher une méthode (par exemple PUT) de modifier par inadvertance une ressource existante lorsque le client pense que la ressource n'existe plus.

À titre de cas particulier, la valeur "*" correspond à toute entité en cours de la ressource.

```
If-None-Match = "If-None-Match" ":" ( "*" | 1#étiquette d'entité )
```

Si une 'étiquette d'entité correspond à l'étiquette d'entité de l'entité qui aurait été retournée dans la réponse à une demande GET similaire (sans l'en-tête If-None-Match) sur cette ressource, ou si "*" est donné et qu'aucune entité en cours n'existe pour cette ressource, le serveur NE DOIT PAS alors effectuer la méthode demandée, sauf s'il est exigé de le faire parce que la date de modification de la ressource ne correspond pas à celle fournie dans un champ d'en-tête If-Modified-Since dans la demande. Au lieu de cela, si la méthode de demande était GET ou HEAD, le serveur DEVRAIT répondre par une réponse 304 (Non modifiée) incluant les champs d'en-tête relatifs à l'antémémoire (en particulier ETag) d'une des entités qui correspondaient. Pour toutes les autres méthodes de demande, le serveur DOIT répondre par un état de 412 (Échec de précondition).

Voir au paragraphe 13.3.3 les règles pour déterminer si deux étiquettes d'entités correspondent. La fonction de comparaison faible ne peut être utilisé qu'avec des demandes GET ou HEAD.

Si aucune des étiquettes d'entité ne correspond, le serveur PEUT alors effectuer la méthode demandée comme si le champ d'en-tête If-None-Match n'existait pas, mais DOIT aussi ignorer tous les champs d'en-tête If-Modified-Since dans la demande. C'est-à-dire que si aucune étiquette d'entité ne correspond, le serveur NE DOIT PAS retourner une réponse 304 (Non modifiée).

Si, en l'absence du champ d'en-tête If-None-Match, la demande résulterait en autre chose qu'un état 2xx ou 304, l'en-tête If-None-Match DOIT alors être ignoré. (Voir au paragraphe 13.3.4 l'exposé sur le comportement du serveur lorsque If-Modified-Since et If-None-Match apparaissent dans la même demande.)

La signification de "If-None-Match: *" est que la méthode NE DOIT PAS être effectuée si la représentation choisie par le serveur d'origine (ou par une antémémoire, éventuellement en utilisant le mécanisme Vary, voir au paragraphe 14.44) existe, et DEVRAIT être effectuée si la représentation n'existe pas. Ce dispositif est destiné à empêcher les compétitions entre les opérations PUT.

Exemples :

```
If-None-Match: "xyzzy"
If-None-Match: W/"xyzzy"
If-None-Match: "xyzzy", "r2d2xxxx", "c3piozzzz"
If-None-Match: W/"xyzzy", W/"r2d2xxxx", W/"c3piozzzz"
If-None-Match: *
```

Le résultat d'une demande ayant à la fois un champ d'en-tête If-None-Match et un champ d'en-tête If-Match ou If-Unmodified-Since n'est pas défini dans la présente spécification.

14.27 If-Range

Si un client a une copie partielle d'une entité dans son antémémoire, et souhaite avoir une copie à jour de l'entité entière dans son antémémoire, il peut utiliser l'en-tête de demande Range avec un GET conditionnel (utilisant If-Unmodified-Since et/ou If-Match.) Cependant, si la condition échoue parce que l'entité a été modifiée, le client devrait alors faire une seconde demande pour obtenir le corps d'entité en cours entier.

L'en-tête If-Range permet à un client de "court-circuiter" la seconde demande. De façon informelle, sa signification est "si l'entité est inchangée, m'envoyer la ou les parties qui me manquent ; autrement, m'envoyer la nouvelle entité entière".

```
If-Range = "If-Range" ":" ( étiquette d'entité | HTTP-date )
```

Si le client n'a pas d'étiquette d'entité pour une entité, mais qu'il a une date Last-Modified, il PEUT utiliser cette date dans un en-tête If-Range. (Le serveur peut distinguer entre une date HTTP valide et toute forme d'étiquette d'entité en examinant seulement deux caractères.) L'en-tête If-Range ne DEVRAIT être utilisé qu'avec un en-tête Range, et DOIT être ignoré si la demande n'inclut pas d'en-tête Range, ou si le serveur ne prend pas en charge l'opération de sous-gamme.

Si l'étiquette d'entité donnée dans l'en-tête If-Range correspond à l'étiquette d'entité en cours pour l'entité, le serveur DEVRAIT alors fournir la sous-gamme spécifiée de l'entité en utilisant une réponse 206 (Contenu partiel). Si l'étiquette d'entité ne correspond pas, le serveur DEVRAIT alors retourner l'entité entière en utilisant une réponse 200 (OK).

14.28 If-Unmodified-Since

Le champ d'en-tête de demande If-Unmodified-Since *(si non modifiée depuis le)* est utilisé avec une méthode pour le rendre conditionnel. Si la ressource demandée n'a pas été modifiée depuis l'heure spécifiée dans ce champ, le serveur DEVRAIT effectuer l'opération demandée comme si l'en-tête If-Unmodified-Since n'était pas présent.

Si la variante demandée a été modifiée depuis l'heure spécifiée, le serveur NE DOIT PAS effectuer l'opération demandée, et DOIT retourner une réponse 412 (Échec de pré-condition).

```
If-Unmodified-Since = "If-Unmodified-Since" ":" HTTP-date
```

Un exemple de ce champ est :

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

Si la demande devait normalement (c'est-à-dire, sans l'en-tête If-Unmodified-Since) résulter en n'importe quoi d'autre qu'un état 2xx ou 412, l'en-tête If-Unmodified-Since DEVRAIT être ignoré.

Si la date spécifiée est invalide, l'en-tête est ignoré.

Le résultat d'une demande ayant à la fois un champ d'en-tête If-Unmodified-Since et un champ d'en-tête If-None-Match ou If-Modified-Since est non défini dans la présente spécification.

14.29 Last-Modified

Le champ d'en-tête d'entité Last-Modified indique la date et l'heure à laquelle le serveur d'origine estime que la variante a été modifiée pour la dernière fois.

```
Last-Modified = "Last-Modified" ":" HTTP-date
```

Un exemple de son utilisation est :

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

La signification exacte de ce champ d'en-tête dépend de la mise en œuvre du serveur d'origine et de la nature de la ressource d'origine. Pour des fichiers, cela peut être simplement l'heure de la dernière modification du

système de fichier. Pour des entités avec des parties incluses de façon dynamique, cela peut être la plus récente de l'ensemble des dernières heures de modification de ses parties composantes. Pour des passerelles de base de données, cela peut être l'horodatage de la dernière mise à jour de l'enregistrement. Pour des objets virtuels, cela peut être l'heure du dernier changement d'état interne.

Un serveur d'origine NE DOIT PAS envoyer une date Last-Modified postérieure à l'heure de l'origine du message du serveur. Dans de tels cas, où la dernière modification de la ressource indiquerait une heure située dans le futur, le serveur DOIT remplacer cette date par la date d'origine du message.

Un serveur d'origine DEVRAIT obtenir une valeur Last-Modified de l'entité aussi proche que possible de l'heure à laquelle il génère la valeur Date de sa réponse. Ceci permet au receveur de faire une estimation précise de l'heure de modification de l'entité, en particulier si l'entité change à peu de temps de la génération de la réponse.

Les serveurs HTTP/1.1 DEVRAIENT envoyer Last-Modified chaque fois que c'est faisable.

14.30 Location

Le champ d'en-tête de réponse Location est utilisé pour rediriger le receveur sur une localisation autre que l'URI de demande pour achever la demande ou identifier une nouvelle ressource. Pour les réponses 201 (Créé), la localisation est celle de la nouvelle ressource qui a été créée par la demande. Pour les réponses 3xx, la localisation DEVRAIT indiquer l'URI préféré du serveur pour une redirection automatique sur la ressource. La valeur du champ consiste en un seul URI absolu.

```
Location       = "Location" ":" absoluteURI
```

Exemple :

```
Location: http://www.w3.org/pub/WWW/People.html
```

Note Le champ d'en-tête Content-Location (paragraphe 14.14) diffère de Location en ce que Content-Location identifie la localisation d'origine de l'entité enclose dans la demande. Il est donc possible qu'une réponse contienne les champs d'en-tête à la fois de Location et Content-Location. Voir aussi au paragraphe 13.10 les exigences d'antémémoire de certaines méthodes.

14.31 Max-Forwards

Le champ d'en-tête de demande Max-Forwards *(transmissions maximales)* fournit un mécanisme avec les méthodes TRACE (paragraphe 9.8) et OPTIONS (paragraphe 9.2) pour limiter le nombre de mandataires ou de passerelles qui peuvent transmettre la demande au prochain serveur entrant. Ceci peut être utile lorsque le client essaye de suivre la trace d'une chaîne de demande qui paraît être défaillante ou boucler à mi-chaîne.

```
Max-Forwards   = "Max-Forwards" ":" 1*DIGIT
```

La valeur de Max-Forwards est un entier décimal qui indique le nombre restant de fois que ce message de demande peut être transmis.

Chaque mandataire ou passerelle receveur d'une demande TRACE ou OPTIONS qui contient un champ d'en-tête Max-Forwards DOIT vérifier et mettre à jour sa valeur avant de retransmettre la demande. Si la valeur reçue est zéro (0), le receveur NE DOIT PAS transmettre la demande ; il DOIT à la place répondre comme receveur final. Si la valeur Max-Forwards reçue est supérieure à zéro, le message transmis DOIT alors contenir un champ Max-Forwards mis à jour avec une valeur décrémentée de un (1).

Le champ d'en-tête Max-Forwards PEUT être ignoré pour toutes les autres méthodes définies par la présente spécification et pour toute méthode d'extension pour laquelle il n'est pas fait explicitement référence à cette définition de méthode.

14.32 Pragma

Le champ d'en-tête général Pragma est utilisé pour inclure des directives spécifiques d'une mise en œuvre qui

pourraient s'appliquer à tout receveur le long de la chaîne des demandes/réponses. Toutes les directives Pragma spécifient un comportement facultatif du point de vue du protocole ; cependant, certains systèmes PEUVENT exiger que ce comportement soit cohérent avec les directives.

```
Pragma  = "Pragma" ":" 1#pragma-directive
pragma-directive  = "no-cache" | extension-pragma
extension-pragma  = token [ "=" ( token | quoted-string ) ]
```

Lorsque la directive no-cache *(pas d'antémémoire)* est présente dans un message de demande, une application DEVRAIT transmettre la demande vers le serveur d'origine même si il a une copie en antémémoire de ce qui est demandé. Cette directive Pragma a la même sémantique que la directive no-cache (voir au paragraphe 14.9) et elle est définie ici pour la rétro-compatibilité avec HTTP/1.0. Les clients DEVRAIENT inclure les deux champs d'en-tête lorsqu'une demande no-cache est envoyée à un serveur dont on ne sait pas s'il est conforme à HTTP/1.1.

Les directives Pragma DOIVENT être passées par un mandataire ou passerelle d'application, quelle que soit leur signification pour cette application, car les directives pourraient être applicables à tous les receveurs le long de la chaîne des demandes/réponses. Il n'est pas possible de spécifier une directive pragma pour un receveur spécifique ; cependant, toute directive pragma non pertinente pour un receveur DEVRAIT être ignorée par ce receveur.

Les antémémoires HTTP/1.1 DEVRAIENT traiter "Pragma: no-cache" comme si le client avait envoyé "Cache-Control: no-cache". Aucune nouvelle directive Pragma ne sera définie dans HTTP.

Note Comme la signification de "Pragma: no-cache" comme champ d'en-tête de réponse n'est pas réellement spécifiée, elle ne fournit pas un remplacement fiable pour "Cache-Control: no-cache" dans une réponse.

14.33 Proxy-Authenticate

Le champ d'en-tête de réponse Proxy-Authenticate DOIT être inclus au titre d'une réponse 407 (Authentification du mandataire exigée). La valeur du champ consiste en une sommation d'indiquer le schéma d'authentification et les paramètres applicables au mandataire pour cet URI de demande.

```
Proxy-Authenticate  = "Proxy-Authenticate" ":" 1#challenge
```

Le processus d'authentification d'accès HTTP est décrit dans "Authentification HTTP : Authentification d'accès de base et par résumé" [43]. À la différence de WWW-Authenticate, le champ d'en-tête Proxy-Authenticate ne s'applique qu'à la connexion en cours et NE DEVRAIT PAS être passé aux clients vers l'aval. Cependant, un mandataire intermédiaire pourrait avoir besoin d'obtenir ses propres accréditifs en les demandant au client aval, ce qui dans certaines circonstances apparaîtra si le mandataire retransmet le champ d'en-tête Proxy-Authenticate.

14.34 Proxy-Authorization

Le champ d'en-tête de demande Proxy-Authorization permet au client de s'identifier (ou son utilisateur) auprès d'un mandataire qui exige l'authentification. La valeur du champ Proxy-Authorization consiste en accréditifs qui contiennent les informations d'authentification de l'agent d'utilisateur pour le mandataire et/ou le domaine des ressources demandées.

```
Proxy-Authorization  = "Proxy-Authorization" ":" credentials
```

Le processus d'authentification d'accès HTTP est décrit dans "Authentification HTTP : Authentification d'accès de base et par résumé" [43]. À la différence de Authorization, le champ d'en-tête Proxy-Authorization ne s'applique qu'au prochain mandataire sortant qui a demandé l'authentification en utilisant le champ Proxy-Authenticate. Lorsque plusieurs mandataires sont utilisés dans une chaîne, le champ d'en-tête Proxy-Authorization est consommé par le premier mandataire sortant qui attendait de recevoir des accréditifs. Un mandataire PEUT relayer les accréditifs provenant de la demande du client jusqu'au prochain mandataire si c'est le mécanisme par lequel les mandataires authentifient de façon coopérative une demande donnée.

14.35 Gamme

14.35.1 Gamme d'octets

Comme toutes les entités HTTP sont représentées dans les messages HTTP comme des séquences d'octets, le concept de gamme d'octets est significatif pour toute entité HTTP. (Cependant, tous les clients et serveurs n'ont pas besoin de prendre en charge les opérations de gamme d'octets.)

Les spécifications de gamme d'octets dans HTTP s'appliquent à la séquence d'octets dans le corps d'entité (pas nécessairement la même que le corps de message).

Une opération de gamme d'octet PEUT spécifier une seule gamme d'octets ou un ensemble de gammes au sein d'une seule entité.

```
ranges-specifier  = byte-ranges-specifier
byte-ranges-specifier  = bytes-unit "=" byte-range-set
byte-range-set  = 1#( byte-range-spec | suffix-byte-range-spec )
byte-range-spec  = first-byte-pos "-" [last-byte-pos]
first-byte-pos  = 1*DIGIT
last-byte-pos  = 1*DIGIT
```

La valeur first-byte-pos dans un byte-range-spec *(spécification de gamme d'octet)* donne le décalage d'octet dans le premier octet d'une gamme. La valeur last-byte-pos donne le décalage d'octet dans le dernier octet de la gamme ; c'est-à-dire que les positions d'octet spécifiées sont inclusives. Les décalages d'octet commencent à zéro.

Si la valeur de last-byte-pos est présente, elle DOIT être supérieure ou égale à la valeur de first-byte-pos dans cette spécification de gamme d'octets ; sinon la byte-range-spec est syntaxiquement invalide. Le receveur d'un byte-range-set qui inclut une ou plusieurs valeurs de byte-range-spec syntaxiquement invalides DOIT ignorer le champ d'en-tête qui inclut ce byte-range-set.

Si la valeur last-byte-pos est absente, ou si la valeur est supérieure ou égale à la longueur actuelle du corps d'entité, last-byte-pos est pris comme égal à un de moins que la longueur actuelle en octets du corps d'entité.

Par son choix de last-byte-pos, un client peut limiter le nombre d'octets restitués sans connaître la taille de l'entité.

```
suffix-byte-range-spec = "-" suffix-length
suffix-length = 1*DIGIT
```

Un suffix-byte-range-spec *(spécification de suffixe de gamme d'octet)* est utilisé pour spécifier le suffixe du corps d'entité, d'une longueur donnée par la valeur de suffix-length. (C'est-à-dire que cette forme spécifie les N derniers octets d'un corps d'entité.) Si l'entité est plus courte que la longueur de suffixe spécifiée, tout le corps d'entité est utilisé.

Si un byte-range-set syntaxiquement valide inclut au moins un byte-range-spec dont le first-byte-pos est inférieur à la longueur actuelle du corps d'entité, ou d'au moins un suffix-byte-range-spec avec une longueur de suffixe différente de zéro, le byte-range-set peut être satisfait. Autrement, le byte-range-set ne peut pas être satisfait. Si le byte-range-set ne peut pas être satisfait, le serveur DEVRAIT retourner une réponse avec un état de 416 (Gamme demandée impossible à satisfaire). Autrement, le serveur DEVRAIT retourner une réponse avec un état de 206 (Contenu partiel) contenant les gammes qui peuvent être satisfaites pour le corps d'entité.

Exemples de valeurs de byte-ranges-specifier *(spécificateur de gammes d'octets)* (en supposant un corps d'entité d'une longueur de 10000) :

- 500 premiers octets (décalage d'octet de 0 à 499, inclus) : bytes=0-499
- 500 octets suivants (décalage d'octet de 500 à 999, inclus): bytes=500-999
- 500 derniers octets (décalage d'octet de 9500 à 9999, inclus): bytes=-500
  Ou bytes = 9500-
- Les seuls premiers et derniers octets (octets 0 et 9999) : bytes=0-0,-1
- Plusieurs spécifications légales mais non canoniques des 500 seconds octets (décalages d'octet 500 à 999, inclus) :
  - bytes=500-600,601-999

– bytes=500-700,601-999

14.35.2 Demandes de restitution de gamme

Les demandes HTTP de restitution de gamme utilisant les méthodes GET conditionnelle ou inconditionnelle PEUVENT demander une ou plusieurs sous-gammes de l'entité, au lieu de l'entité entière, en utilisant l'en-tête de demande Range, qui s'applique à l'entité retournée comme résultat de la demande :

```
Range = "Range" ":" ranges-specifier
```

Un serveur PEUT ignorer l'en-tête Range. Cependant, les serveurs d'origine HTTP/1.1 et les antémémoires intermédiaires devraient prendre en charge les gammes d'octet lorsque c'est possible, car Range prend en charge une récupération efficace à partir de défaillances partielles de transferts, et prend en charge une récupération partielle efficace de grandes entités.

Si le serveur prend en charge l'en-tête Range et si la ou les gammes spécifiées sont appropriées pour l'entité :

– La présence d'un en-tête Range dans un GET inconditionnel modifie ce qui est retourné si le GET est réussi par ailleurs. En d'autres termes, la réponse porte un code d'état de 206 (Contenu partiel) au lieu de 200 (OK).

– La présence d'un en-tête Range dans un GET conditionnel (une demande utilisant If-Modified-Since et/ou If-None-Match, ou If-Unmodified-Since et/ou If-Match) modifie ce qui est retourné si GET est par ailleurs réussi et si la condition est vraie. Elle n'affecte pas la réponse 304 (Non modifiée) retournée si la condition est fausse.

Dans certains cas, il peut être plus approprié d'utiliser l'en-tête If-Range (voir au paragraphe 14.27) en plus de l'en-tête Range.

Si un mandataire qui prend en charge les gammes reçoit une demande Range, transmet la demande à un serveur entrant, et reçoit une entité entière en réponse, il DEVRAIT seulement retourner la gamme demandée à son client. Il DEVRAIT mémoriser toute la réponse reçue dans son antémémoire si c'est cohérent avec sa politique d'allocation d'antémémoire.

14.36 Referer

Le champ d'en-tête de demande Referer [sic] permet au client de spécifier, pour le bénéfice du serveur, l'adresse (URI) de la ressource de laquelle l'URI de demande a été obtenu (le "referrer", bien que le champ d'en-tête soit mal épelé.) L'en-tête de demande Referer permet à un serveur de générer des listes de liaisons de retour vers des ressources présentant un intérêt, la constitution d'un enregistrement de journalisation, une mise en antémémoire optimisée, etc. Il permet aussi que les liaisons obsolètes ou mal tapées soient retrouvées pour la maintenance. Le champ Referer NE DOIT PAS être envoyé si l'URI de demande a été obtenu d'une source qui n'a pas son propre URI, comme une entrée du clavier de l'utilisateur.

```
Referer        = "Referer" ":" ( absoluteURI | relativeURI )
```

Exemple :

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

Si la valeur du champ est un URI relatif, il DEVRAIT être interprété par rapport à l'URI de demande. L'URI NE DOIT PAS inclure un fragment. Voir au paragraphe 15.1.3 les considérations sur la sécurité.

14.37 Retry-After

Le champ d'en-tête de réponse Retry-After *(réessayer dans)* peut être utilisé avec une réponse 503 (Service indisponible) pour indiquer pendant combien de temps on pense que le service va être indisponible pour le client demandeur. Ce champ PEUT aussi être utilisé avec toute réponse 3xx (Redirection) pour indiquer la durée minimum pendant laquelle l'agent d'utilisateur va devoir attendre avant de produire la demande redirigée.

La valeur de ce champ peut être une date HTTP ou un nombre entier de secondes (en décimal) après l'heure de la réponse.

```
Retry-After  = "Retry-After" ":" ( HTTP-date | delta-secondes )
```

Voici deux exemples de son utilisation :

```
Retry-After: Fri, 31 Dec 1999 23:59:59 GMT
Retry-After: 120
```

Dans le dernier exemple, le délai est de 2 minutes.

14.38 Server

Le champ d'en-tête de réponse Server contient des informations sur le logiciel utilisé par le serveur d'origine pour traiter la demande. Le champ peut contenir plusieurs jetons de produits (paragraphe 3.8) et des commentaires identifiant le serveur et tout sous-produit significatif. Les jetons de produits figurent sur une liste dans l'ordre de leur signification pour identifier l'application.

```
Server  = "Server" ":" 1*( product | comment )
```

Exemple :

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

Si la réponse doit être transmise à travers un mandataire, l'application mandataire NE DOIT PAS modifier l'en-tête de réponse Server. Elle DEVRAIT plutôt inclure un champ Via (comme décrit au paragraphe 14.45).

Note Révéler la version spécifique du logiciel du serveur peut amener la machine serveur à être plus vulnérable aux attaques contre le logiciel qui est connu pour avoir des failles dans sa sécurité. Les mises en œuvre de serveur sont invitées à faire de ce champ une option configurable.

14.39 TE

Le champ d'en-tête de demande TE indique quels codages d'extension de transfert il acceptera dans la réponse et si il veut ou non accepter des champs de queue dans un codage de transfert fragmenté. Sa valeur peut être constituée du mot clé "trailers" et/ou d'une liste séparée par des virgules de noms de codage d'extension de transfert avec des paramètres d'acceptation facultatifs (comme décrit au paragraphe 3.6).

```
TE        = "TE" ":" #( t-codings )
t-codings = "trailers" | ( transfer-extension [ accept-params ] )
```

La présence du mot clé "trailers" indique que le client est d'accord pour accepter les champs d'en-queue dans un codage de transfert fragmenté, comme défini au paragraphe 3.6.1. Ce mot clé est réservé à l'utilisation avec les valeurs de codage de transfert même si il ne représente pas lui-même un codage de transfert.

Des exemples de cette utilisation sont :

```
TE: deflate
TE:
TE: trailers, deflate;q=0.5
```

Le champ d'en-tête TE ne s'applique qu'à la connexion immédiate. Donc, le mot clé DOIT être fourni au sein d'un champ d'en-tête Connection (paragraphe 14.10) chaque fois que TE est présent dans un message HTTP/1.1.

Un serveur vérifie si un codage de transfert est acceptable, conformément au champ TE, en utilisant ces règles :

1. Le codage de transfert "tronqué" est toujours acceptable. Si le mot clé "trailers" figure sur la liste, le client indique qu'il est d'accord pour accepter les champs d'en-queue dans la réponse fragmentée au

nom de lui-même et de tout client aval. Cela implique que, s'il est donné, le client déclare que tous les clients vers l'aval sont d'accord pour accepter les champs d'en-queue dans la réponse transmise, ou qu'il va essayer de mettre la réponse en mémoire tampon au nom des receveurs de l'aval.

Note : HTTP/1.1 ne définit aucun moyen de limiter la taille d'une réponse fragmentée de sorte qu'un client peut être assuré de mettre en mémoire tampon la réponse entière.

2. Si le codage de transfert testé est un des codages de transfert de la liste du champ TE, il est alors acceptable sauf s'il est accompagné d'une qvalue de 0. (Comme défini au paragraphe 3.9, une qvalue de 0 signifie "non acceptable.")

3. Si plusieurs codages de transfert sont acceptables, le codage de transfert acceptable avec la plus forte qvalue différente de zéro est préféré. Le codage de transfert "fragmenté" a toujours une qvalue de 1.

Si le champ TE est vide ou si aucun champ TE n'est présent, seul le codage de transfert est "fragmenté". Un message sans codage de transfert est toujours acceptable.

14.40 Trailer

La valeur de champ générale Trailer indique que l'ensemble donné de champs d'en-tête est présent dans la queue d'un message codé avec un codage de transfert fragmenté.

```
Trailer  = "Trailer" ":" 1#field-name
```

Un message HTTP/1.1 DEVRAIT inclure un champ d'en-tête Trailer dans un message utilisant un codage de transfert fragmenté avec une queue non vide. Faire ainsi permet au receveur de savoir quels champs d'en-tête attendre dans la queue.

Si aucun champ d'en-tête Trailer n'est présent, la queue NE DEVRAIT PAS inclure de champ d'en-tête. Voir au paragraphe 3.6.1 les restrictions sur l'utilisation des champs de queue dans un codage de transfert "fragmenté". Les champs d'en-tête de message figurant sur la liste du champ d'en-tête Trailer NE DOIVENT PAS inclure les champs d'en-tête suivants :

- Transfer-Encoding
- Content-Length
- Trailer

14.41 Transfer-Encoding

Le champ d'en-tête général Transfer-Encoding *(codage de transfert)* indique (s'il en est) quel type de transformation a été appliquée au corps de message afin de le transférer en toute sécurité entre l'envoyeur et le receveur. Ceci diffère du codage de contenu en ce que le codage de transfert est une propriété du message, non de l'entité.

```
Transfer-Encoding   "Transfer-Encoding" ":" 1#transfer-coding
```

Les codages de transfert sont définis au paragraphe 3.6. Un exemple en est :

```
Transfer-Encoding: chunked
```

Si plusieurs codages ont été appliqués à une entité, la liste des codages de transfert DOIT être dans l'ordre dans lequel ils ont été appliqués. Des informations supplémentaires sur les paramètres de codage PEUVENT être fournis par d'autres champs d'en-tête d'entité non définis par la présente spécification.

De nombreuses applications HTTP/1.0 plus anciennes ne comprennent pas l'en-tête Transfer-Encoding.

14.42 Upgrade

L'en-tête général Upgrade *(mettre à niveau)* permet au client de spécifier quels protocoles de communication supplémentaires il prend en charge et aimerait utiliser si le serveur trouve approprié de passer à ces protocoles.

Le serveur DOIT utiliser le champ d'en-tête Upgrade au sein d'une réponse 101 (Changement de protocoles) pour indiquer quel ou quels protocoles seront substitués.

```
Upgrade  = "Upgrade" ":" 1#product
```

Par exemple,

```
Upgrade: HTTP/2.0, SHTTP/1.3, IRC/6.9, RTA/x11
```

Le champ d'en-tête Upgrade est destiné à fournir un mécanisme de transition simple de HTTP/1.1 à un autre protocole incompatible. Il le fait en permettant au client de faire connaître son désir d'utiliser un autre protocole, comme une version plus récente de HTTP avec un numéro de version majeure supérieur, bien que la demande en cours ait été faite en utilisant HTTP/1.1. Ceci facilite la difficile transition entre des protocoles incompatibles en permettant au client d'initier une demande dans le protocole le plus couramment utilisé tout en indiquant au serveur qu'il aimerait utiliser un "meilleur" protocole, si disponible (où "meilleur" est déterminé par le serveur, éventuellement conformément à la nature de la méthode et/ou ressource demandée).

Le champ d'en-tête Upgrade ne s'applique qu'aux changements de protocole de couche application sur la connexion de couche transport existante. Upgrade ne peut pas être utilisé pour insister sur un changement de protocole ; son acceptation et son utilisation par le serveur est facultative. Les capacités et la nature de la communication de couche application après le changement de protocole dépendent entièrement du nouveau protocole choisi, bien que la première action après le changement de protocole DOIVE être une réponse à la demande HTTP initiale qui contenait le champ d'en-tête Upgrade.

Le champ d'en-tête Upgrade ne s'applique qu'à la connexion immédiate. Donc, le mot clé Upgrade DOIT être fourni dans un champ d'en-tête Connection (paragraphe 14.10) chaque fois que Upgrade est présent dans un message HTTP/1.1.

Le champ d'en-tête Upgrade ne peut pas être utilisé pour indiquer le passage à un protocole sur une connexion différente. A cette fin, il est plus approprié d'utiliser une réponse de redirection 301, 302, 303, ou 305.

La présente spécification ne définit que le nom de protocole "HTTP" pour être utilisé par la famille des protocoles de transfert Hypertexte, comme défini par les règles de version HTTP du paragraphe 3.1 et les futures mises à jour de la présente spécification. Tout jeton peut être utilisé comme nom de protocole ; cependant, il ne sera utile que si le client et le serveur associent tous deux le nom au même protocole.

14.43 User-Agent

Le champ d'en-tête de demande User-Agent contient des informations sur l'agent d'utilisateur d'origine de la demande. Il sert à des fins statistiques, pour le traçage des violations de protocole, et pour la reconnaissance automatisée des agents d'utilisateur afin de tailler sur mesure des réponses et éviter des limitations d'agents d'utilisateur particuliers. Les agents d'utilisateur DEVRAIENT inclure ce champ avec les demandes. Le champ peut contenir plusieurs jetons de produit (paragraphe 3.8) et des commentaires identifiant l'agent et tous sous-produits qui forment une part significative de l'agent d'utilisateur. Par convention, la liste des jetons de produit figure dans l'ordre de leur signification pour identifier l'application.

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

Exemple :

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

14.44 Vary

La valeur de champ Vary indique l'ensemble des champs d'en-tête de demande qui déterminent pleinement, alors que la réponse est fraîche, si une antémémoire est autorisée à utiliser la réponse pour répondre à une demande ultérieure sans revalidation. Pour les réponses qui ne peuvent pas être mises en antémémoire ou qui sont périmées, la valeur de champ Vary informe l'agent d'utilisateur sur les critères qui ont été utilisés pour choisir la représentation. Une valeur de champ Vary de "*" implique qu'une antémémoire ne peut pas déterminer à partir des en-têtes de la demande d'une demande ultérieure si cette réponse est la représentation

appropriée. Voir au paragraphe 13.6 l'utilisation du champ d'en-tête Vary par les antémémoires.

```
Vary   = "Vary" ":" ( "*" | 1#field-name )
```

Un serveur HTTP/1.1 DEVRAIT inclure un champ d'en-tête Vary avec toute réponse éligible à la mise en antémémoire qui est soumise à une négociation conduite par le serveur. Le faire permet à une antémémoire d'interpréter de façon appropriée des demandes futures sur cette ressource et d'informer l'agent d'utilisateur de la présence de négociations sur cette ressource. Un serveur PEUT inclure un champ d'en-tête Vary avec une réponse non metteble en antémémoire qui est soumise à négociation conduite par le serveur, car cela peut fournir à l'agent d'utilisateur des informations utiles sur les dimensions dans lesquelles varie la réponse au moment de la réponse.

Une valeur de champ Vary consistant en une liste de noms de champs signale que la représentation choisie pour la réponse est fondée sur un algorithme de choix qui ne considère QUE les valeurs de champ d'en-tête de demande listées en choisissant la représentation la plus appropriée. Une antémémoire PEUT supposer que le même choix sera fait pour des demandes futures avec les mêmes valeurs pour les noms de champ listés, pour la durée pendant laquelle la réponse est fraîche.

Les noms de champ donnés ne sont pas limités à l'ensemble des champs d'en-tête de demande standard définis par la présente spécification. Les noms de champ sont insensibles à la casse.

Une valeur de champ Vary de "*" signale que des paramètres non spécifiés non limités aux en-têtes de demande (par exemple, l'adresse réseau du client) jouent un rôle dans le choix de la représentation de la réponse. La valeur "*" NE DOIT PAS être générée par un serveur mandataire ; elle ne peut être générée que par un serveur d'origine.

14.45 Via

Le champ d'en-tête général Via DOIT être utilisé par les passerelles et mandataires pour indiquer les protocoles et receveurs intermédiaires entre l'agent d'utilisateur et le serveur sur les demandes, et entre le serveur d'origine et le client sur les réponses. C'est analogue au champ "Received" de la RFC 822 [9] et est destiné à être utilisé pour suivre les messages vers l'avant, en évitant les boucles de demandes, et en identifiant les capacités de protocole de tous les envoyeurs le long de la chaîne des demandes/réponses.

```
Via = "Via" ":" 1#( received-protocol received-by [ comment ] )
received-protocol = [ protocol-name "/" ] protocol-version
protocol-name = jeton
protocol-version = jeton
received-by = ( hôte [ ":" port ] ) | pseudonyme
pseudonyme = jeton
```

Le received-protocol indique la version de protocole du message reçu par le serveur ou client le long de chaque segment de la chaîne des demandes/réponses. La version de received-protocol est ajoutée à la valeur du champ Via lorsque le message est transmis de sorte que les informations sur les capacités du protocole des applications vers l'amont restent visibles pour tous les receveurs.

Le protocol-name n'est facultatif que si et seulement si il devrait être "HTTP". Le champ received-by est normalement l'hôte et le numéro de port facultatif du serveur ou client receveur qui transmet ensuite le message. Cependant, si l'hôte réel est considéré comme une information sensible, il PEUT être remplacé par un pseudonyme. Si le port n'est pas donné, il PEUT être supposé être le port par défaut du received-protocol.

Des valeurs de champ Via multiples représentent chaque mandataire ou passerelle qui a transmis le message. Chaque receveur DOIT ajouter ses informations de telle sorte que le résultat final soit ordonné conformément à la séquence des applications de transmission.

Les commentaires PEUVENT être utilisés dans le champ d'en-tête Via pour identifier le logiciel du mandataire ou passerelle qui reçoit, comme dans les champs d'en-tête User-Agent et Server. Cependant, tous les commentaires dans le champ Via sont facultatifs et PEUVENT être retirés par tout receveur avant la transmission du message.

Par exemple, un message de demande pourrait être envoyé d'un agent d'utilisateur HTTP/1.0 à un mandataire interne du nom de code "fred", qui utilise HTTP/1.1 pour transmettre la demande à un mandataire public à

nowhere.com, qui termine la demande en l'envoyant au serveur d'origine à www.ics.uci.edu. La demande reçue par www.ics.uci.edu aurait alors le champ d'en-tête V Ia suivant :

```
Via: 1.0 fred, 1.1 nowhere.com (Apache/1.1)
```

Les mandataires et passerelles utilisés comme portail à travers un pare-feu de réseau NE DEVRAIENT PAS, par défaut, transmettre les noms et ports des hôtes au sein de la région du pare-feu. Ces informations NE DEVRAIENT être propagées que si elles sont explicitement activées. Si elles ne sont pas activées, l'hôte received-by de tout hôte derrière le pare-feu DEVRAIT être remplacé par un pseudonyme approprié pour cet hôte.

Pour les organisations qui ont de fortes exigences de confidentialité en matière de dissimulation des structures internes, un mandataire PEUT combiner une sous-séquence ordonnée d'entrées de champ d'en-tête Via avec des valeurs identiques de received-protocol en une seule de ces entrées. Par exemple,

```
Via: 1.0 ricky, 1.1 ethel, 1.1 fred, 1.0 lucy
```

pourrait devenir

```
Via: 1.0 ricky, 1.1 mertz, 1.0 lucy
```

Les applications NE DEVRAIENT PAS combiner plusieurs entrées sauf si elles sont sous le même contrôle organisationnel et si les hôtes ont déjà été remplacés par des pseudonymes. Les applications NE DOIVENT PAS combiner des entrées qui ont des valeurs de received-protocol différentes.

14.46 Warning

Le champ d'en-tête général Warning est utilisé pour porter des informations supplémentaires sur l'état ou la transformation d'un message qui pourrait n'être pas reflété dans le message. Ces informations sont normalement utilisées pour avertir d'un possible manque de la transparence sémantique d'opérations de mise en antémémoire ou transformations appliquées au corps d'entité du message.

Les en-têtes d'avertissement sont envoyés avec des réponses utilisant :

```
Warning    = "Warning" ":" 1#warning-value
warning-value = warn-code SP warn-agent SP warn-text [SP warn-date]
warn-code  = 3DIGIT
warn-agent = ( hôte [ ":" port ] ) | pseudonyme
  ; le nom ou pseudonyme du serveur qui ajoute l'en-tête Warning, pour un débogage
warn-text  = quoted-string
warn-date  = <"> HTTP-date <">
```

Une réponse PEUT porter plus d'un en-tête Warning.

Le warn-text DEVRAIT être en langage naturel et jeu de caractères qui soient très vraisemblablement intelligibles à l'utilisateur humain qui reçoit la réponse. Cette décision PEUT être fondée sur toute connaissance disponible, telles que la localisation de l'antémémoire ou de l'utilisateur, le champ Accept-Language dans une demande, le champ Content-Language dans une réponse, etc. Le langage par défaut est l'anglais et le jeu de caractères par défaut est ISO-8859-1.

Si un jeu de caractères autre que ISO-8859-1 est utilisé, il DOIT être codé dans le warn-text en utilisant la méthode décrite dans la RFC 2047 [14].

Les en-têtes Warning peuvent en général être appliqués à tout message, cependant, certains warn-codes sont spécifiques des antémémoires et ne peuvent être appliqués qu'aux messages de réponse. De nouveaux en-têtes Warning DEVRAIENT être ajoutés après tout en-tête Warning existant. Une antémémoire NE DOIT PAS supprimer un en-tête Warning qu'elle a reçu avec un message. Cependant, si une antémémoire valide avec succès une entrée d'antémémoire, elle DEVRAIT retirer tous les en-têtes Warning attachés précédemment à cette entrée excepté comme spécifié pour les codes Warning spécifiques. Elle DOIT alors ajouter tous les en-têtes Warning reçus dans la réponse de validation. En d'autres termes, les en-têtes Warning sont ceux qui devraient être attachés à la réponse pertinente la plus récente.

Lorsque plusieurs en-têtes Warning sont attachés à une réponse, l'agent d'utilisateur devrait informer l'utilisateur d'autant qu'il est possible, dans l'ordre où ils apparaissent dans la réponse. Si il n'est pas possible d'informer l'utilisateur de tous les avertissements, l'agent d'utilisateur DEVRAIT suivre cette heuristique :

- Les avertissements qui apparaissent tôt dans la réponse prennent le pas sur ceux qui y apparaissent tard.
- Les avertissements dans le jeu de caractères préféré de l'utilisateur prennent le pas sur les avertissements dans d'autres jeux de caractères mais avec des warn-codes et warn-agents identiques.

Les systèmes qui génèrent plusieurs en-têtes d'avertissement DEVRAIENT les ordonner en gardant en mémoire ce comportement d'agent d'utilisateur.

Les exigences pour le comportement des antémémoires par rapport aux avertissements figurent au paragraphe 13.1.2.

Une liste des codes d'avertissement actuellement définis, avec chacun un texte d'avertissement recommandé en français, et une description de sa signification est donnée ci-dessous.

110 Réponse périmée
DOIT être inclus chaque fois que la réponse retournée est périmée.

111 Échec de revalidation
DOIT être inclus si une antémémoire retourne une réponse périmée à cause de l'échec d'un essai de revalidation de la réponse, due à l'incapacité à atteindre le serveur.

112 Opération déconnectée
DEVRAIT être inclus si l'antémémoire est intentionnellement déconnectée du reste du réseau pendant un temps.

113 Expiration heuristique
DOIT être inclus si l'antémémoire choisit heuristiquement une durée de vie de fraîcheur supérieure à 24 heures et que l'âge de la réponse est supérieur à 24 heures.

199 Avertissements divers
Le texte de l'avertissement PEUT inclure des informations arbitraires à présenter à un utilisateur humain, ou à enregistrer. Un système qui reçoit cet avertissement NE DOIT PAS entreprendre d'action automatique, en dehors de présenter l'avertissement à l'utilisateur.

214 Transformation appliquée
DOIT être ajouté par une antémémoire ou mandataire intermédiaire si elle s'applique à une transformation qui change le codage de contenu (comme spécifié dans l'en-tête Content-Encoding) ou le type de support (comme spécifié dans l'en-tête Content-Type) de la réponse, ou le corps d'entité de la réponse, sauf si ce code d'avertissement apparaît déjà dans la réponse.

299 Avertissements divers persistants
Le texte de l'avertissement PEUT inclure des informations arbitraires à présenter à un utilisateur humain, ou à enregistrer. Un système qui reçoit cet avertissement NE DOIT PAS entreprendre d'action automatique.

Si une mise en œuvre envoie un message avec un ou plusieurs en-têtes Warning dont la version est HTTP/1.0 ou plus ancienne, l'envoyeur DOIT alors inclure dans chaque valeur d'avertissement une warn-date qui corresponde à la date dans la réponse.

Si une mise en œuvre reçoit un message avec une valeur d'avertissement qui inclut une warn-date, et que cette warn-date est différente de la valeur de la date de la réponse, cette valeur d'avertissement DOIT être supprimée du message avant sa mémorisation, transmission ou utilisation. (Ceci empêche les conséquences néfastes de la mise en antémémoire non réfléchie de champs d'en-tête Warning.) Si toutes les valeurs d'avertissement sont supprimées pour cette raison, l'en-tête Warning DOIT être lui aussi supprimé.

14.47 WWW-Authenticate

Le champ d'en-tête de réponse WWW-Authenticate DOIT être inclus dans les messages de réponse 401 (Non

autorisé). La valeur du champ consiste au moins en une mise en cause *(challenge)* qui indique le ou les schémas et paramètres d'authentification applicables à l'URI de demande.

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

Le processus d'authentification d'accès HTTP est décrit dans "Authentification HTTP : Authentification d'accès de base et par résumé" [43]. Il est conseillé aux agents d'utilisateurs d'apporter un soin tout particulier à l'analyse de la valeur du champ WWW-Authenticate car il pourrait contenir plus d'une mise en cause, ou si plus d'un champ d'en-tête WWW-Authenticate est fourni, le contenu d'une mise en cause peut lui-même contenir une liste séparée par des virgules de paramètres d'authentification.

15 Considérations sur la sécurité

La présente section est destinée à informer les développeurs d'applications, les fournisseurs d'informations, et les utilisateurs des limitations de la sécurité dans HTTP/1.1 telles que décrites par le présent document. L'exposé n'inclut pas les solutions définitives des problèmes révélés, bien qu'il fasse quelques suggestions pour réduire les risques pour la sécurité.

15.1 Informations personnelles

Les clients HTTP ont souvent connaissance de grandes quantités d'informations personnelles (par exemple le nom de l'utilisateur, sa localisation, adresse de messagerie électronique, mots de passe, clés de chiffrement, etc.) et DEVRAIENT faire très attention à empêcher les fuites involontaires de telles informations via le protocole HTTP vers d'autres sources. On recommande très fortement de fournir une interface pratique pour que l'utilisateur contrôle la dissémination de telles informations, et que les concepteurs et ceux qui mettent en œuvre soient particulièrement attentifs dans ce domaine. L'histoire montre que des erreurs dans ce domaine créent souvent de sérieux problèmes de sécurité et/ou de confidentialité et génèrent une forte contre publicité pour la société en question.

15.1.1 Abus des informations de journalisation du serveur

Un serveur est en mesure de sauvegarder des données personnelles sur des demandes d'un utilisateur qui pourraient identifier leurs schémas de lecture ou leurs sujets d'intérêt. Ces informations sont clairement de nature confidentielle et leur traitement peut être régi par la loi dans certains pays. Les gens qui utilisent le protocole HTTP pour fournir des données sont responsables si de telles données sont distribuées sans la permission des individus qui sont identifiables par les résultats publiés.

15.1.2 Transfert d'informations sensibles

Comme tout protocole générique de transfert de données, HTTP ne peut pas réguler le contenu des données qui sont transférées, pas plus qu'il n'y a de méthode pour déterminer a priori la sensibilité d'un ensemble d'informations particulier dans le contexte d'une demande donnée. Donc, les applications DEVRAIENT fournir au fournisseur des informations autant de contrôle que possible sur ces informations. Quatre champs d'en-tête méritent une mention particulière dans ce contexte : Server, Via, Referer et From.

La révélation des versions de logiciel spécifiques du serveur peut rendre la machine serveur plus vulnérable aux attaques contre un logiciel qui est connu pour les failles de sa sécurité. Les mises en œuvre DEVRAIENT faire du champ d'en-tête Server une option configurable.

Les mandataires qui servent de portail à travers un pare-feu de réseau DEVRAIENT prendre des précautions particulières en ce qui concerne le transfert des informations d'en-tête qui identifient les hôtes derrière le pare-feu. En particulier, ils DEVRAIENT retirer, ou remplacer par des versions de confiance, tout champ Via généré derrière le pare-feu.

L'en-tête Referer permet que les schémas de lecture soient étudiés et de tirer les liens inverses. Bien que cela puisse être très utile, sa puissance peut être détournée si les détails de l'utilisateur ne sont pas séparés des informations contenues dans le Referer. Même lorsque les informations personnelles ont été retirées, l'en-tête

Referer pourrait indiquer l'URI d'un document privé dont la publication serait inappropriée.

Les informations envoyées dans le champ From peuvent entrer en conflit avec les intérêts de confidentialité de l'utilisateur ou la politique de sécurité de son site, et donc, elles NE DEVRAIENT PAS être transmises sans que l'utilisateur ne soit à même de désactiver, activer, et modifier le contenu du champ. L'utilisateur DOIT être capable d'établir le contenu de ce champ au sein d'une configuration de préférence d'utilisateur ou d'application par défaut.

On suggère, sans l'imposer, qu'une interface d'inversion pratique soit fournie pour que l'utilisateur active ou désactive l'envoi des informations de From et Referer.

Les champs d'en-tête User-Agent (paragraphe 14.43) ou Server (paragraphe 14.38) peuvent parfois être utilisés pour déterminer si un client ou serveur spécifique a une faille particulière dans sa sécurité qui pourrait être exploitée. Malheureusement, ces mêmes informations sont souvent utilisées pour d'autres fins estimables pour lesquelles HTTP n'a pas actuellement de meilleur mécanisme.

15.1.3 Codage des informations sensibles dans l'URI

Comme la source d'un lien peut être une information privée ou peut révéler une source d'informations normalement confidentielle, il est fortement recommandé que l'utilisateur soit capable de choisir si le champ Referer est envoyé ou non. Par exemple, un client logiciel de navigation pourrait avoir un commutateur inverseur pour naviguer ouvertement/anonymement, qui pourrait respectivement activer/désactiver l'envoi des informations de Referer et From.

Les clients NE DEVRAIENT PAS inclure un champ d'en-tête Referer dans une demande (non sécurisée) HTTP si la page de référence a été transférée avec un protocole sécurisé.

Les auteurs de services qui utilisent le protocole HTTP NE DEVRAIENT PAS utiliser des formes fondées sur GET pour la soumission de données sensibles, parce que cela va causer le codage de ces données dans l'URI de demande. De nombreux serveurs, mandataires, et agent d'utilisateurs existants vont enregistrer l'URI de demande dans un endroit où il pourrait être visible à des tiers. Les serveurs peuvent utiliser à la place une soumission de forme fondée sur POST.

15.1.4 Questions de confidentialité en relation avec les en-têtes Accept

Les en-têtes de demande Accept peuvent révéler des informations sur l'utilisateur à tous les serveurs auxquels il accède. L'en-tête Accept-Language peut en particulier révéler des informations que l'utilisateur pourrait considérer comme étant de nature privée, parce que la compréhension de langages particuliers est souvent fortement corrélée à l'appartenance à un groupe ethnique particulier. Les agents d'utilisateur qui offrent l'option de configurer le contenu d'un en-tête Accept-Language à envoyer dans toutes les demandes sont fortement encouragés à laisser le processus de configuration inclure un message qui informe l'utilisateur de la perte de confidentialité que cela implique.

Une approche qui limite la perte de confidentialité serait pour un agent d'utilisateur d'omettre l'envoi des en-têtes Accept-Language par défaut, et de demander à l'utilisateur de commencer ou non l'envoi des en-têtes Accept-Language à un serveur si il détecte, en cherchant tous les champs d'en-tête de réponse Vary générés par le serveur, qu'un tel envoi pourrait améliorer la qualité de service.

Des champs d'en-tête élaborés personnalisés par l'utilisateur, envoyés dans chaque demande, en particulier si ceux-ci comportent des valeurs de qualité, peuvent être utilisés par des serveurs comme identifiants d'utilisateur relativement fiables et de longue durée de vie. De tels identifiants d'utilisateur permettraient aux fournisseurs de contenu de retracer la liste des liens suivis, et permettraient aux fournisseurs de contenus associés de confronter les listes de liens suivis ou les soumissions de forme qui traversent les serveurs pour les utilisateurs individuels. Noter que pour de nombreux utilisateurs qui ne sont pas derrière un mandataire, l'adresse réseau de l'hôte qui fait fonctionner l'agent d'utilisateur va aussi servir comme identifiant d'utilisateur à longue durée. Dans les environnements où les mandataires sont utilisés pour améliorer la confidentialité, les agents d'utilisateur devraient être prudents dans leur offre d'options de configuration d'en-tête Accept aux utilisateurs finaux. Comme mesure de confidentialité extrême, les mandataires pourraient filtrer les en-têtes Accept dans les demandes relayées. Les agents d'utilisateur d'objet général qui fournissent un haut degré de capacité de configuration d'en-tête DEVRAIENT avertir les utilisateurs de la perte de confidentialité qui peut être impliquée.

15.2 Attaques fondées sur les noms de fichier et de voie

Les mises en œuvre de serveurs d'origine HTTP DEVRAIENT veiller à restreindre les documents retournés par les demandes HTTP à ceux qui étaient prévus par les administrateurs du serveur. Si un serveur HTTP traduit des URI HTTP directement en appels de système de fichiers, le serveur DOIT veiller particulièrement à ne pas servir des fichiers dont il n'est pas prévu qu'ils soient livrés à des clients HTTP. Par exemple, UNIX, Microsoft Windows, et autres systèmes d'exploitation utilisent ".." comme composant de chemin pour indiquer un niveau de répertoire supérieur à celui en cours. Sur de tels systèmes, un serveur HTTP DOIT interdire une telle construction dans l'URI de demande car cela permettrait sinon l'accès à une ressource au delà de ce qu'il est entendu être accessible via le serveur HTTP. De même, les fichiers destinés seulement à servir de référence interne au serveur (comme les fichiers de commande d'accès, les fichiers de configuration, et le code d'écriture) DOIVENT être protégés contre une restitution inappropriée, car ils peuvent contenir des informations sensibles. L'expérience a montré que des bogues mineures dans de telles mises en œuvre de serveur HTTP se sont révélées présenter des risques pour la sécurité.

15.3 Usurpation de DNS

Les clients utilisant HTTP s'appuient fortement sur le système des noms de domaine (DNS, *Domain Name Service*) et sont donc généralement sujets à des attaques contre la sécurité fondées sur une mauvaise association délibérée d'adresses IP et de noms DNS. Les clients doivent être prudents lorsqu'ils supposent la validité continue d'une association d'un numéro IP et d'un nom DNS.

En particulier, les clients HTTP DEVRAIENT s'appuyer sur leur résolveur de noms pour confirmer une association d'un numéro IP et d'un nom DNS, plutôt que de mettre en antémémoire le résultat de recherches précédentes de nom d'hôte. De nombreuses plateformes peuvent déjà mettre en antémémoire localement les recherches de nom d'hôte quand c'est approprié, et elles DEVRAIENT être configurées pour le faire. Cependant, Il est approprié que ces recherches ne soient mises en antémémoire que seulement lorsque les informations de durée de vie (TTL, *Time To Live*) rapportées par le serveur de nom font que les informations de l'antémémoire vont rester utiles.

Si les clients HTTP mettent en antémémoire les résultats des recherches de nom d'hôte afin de réaliser une amélioration des performances, ils DOIVENT observer les informations de TTL rapportées par DNS.

Si les clients HTTP n'observent pas ces règles, ils peuvent être mystifiés lorsque change une adresse IP d'un serveur auquel ils ont accédé précédemment. Comme les renumérotages de réseau vont devenir de plus en plus courants [24], la possibilité de cette forme d'attaque va croître. L'observation de cette exigence réduit donc cette vulnérabilité potentielle de la sécurité.

Cette exigence améliore aussi le comportement d'équilibrage de charge des clients pour les serveurs dupliqués utilisant le même nom DNS et réduit la probabilité qu'un utilisateur essuie un échec en accédant à des sites qui utilisent cette stratégie.

15.4 En-tête de localisation et usurpation

Si un seul serveur prend en charge plusieurs organisations qui ne se font pas mutuellement confiance, il DOIT alors vérifier les valeurs des en-têtes Location et Content-Location en réponse car ils sont générés sous la responsabilité des dites organisations, pour s'assurer qu'elles n'essayent pas d'invalider des ressources sur lesquelles elles n'ont pas autorité.

15.5 Problèmes posés par Content-Disposition

La RFC 1806 [35], d'après laquelle est dérivé l'en-tête souvent mis en œuvre Content-Disposition (voir au paragraphe 19.5.1) dans HTTP, pose un certain nombre de problèmes de sécurité très sérieux. Content-Disposition ne fait pas partie de la norme HTTP, mais dans la mesure où il est largement mis en œuvre, on mentionne son utilisation et les risques qu'il fait courir aux mises en œuvre. Voir la RFC 2183 [49] (qui met à jour la RFC 1806) pour les précisions.

15.6 Accréditifs d'authentification et clients inactifs

Les clients et agent d'utilisateurs HTTP existants conservent normalement les informations d'authentification indéfiniment. HTTP/1.1 ne fournit pas de méthode pour qu'un serveur conduise les clients à éliminer ces accréditifs en antémémoire. C'est un défaut significatif qui exige à l'avenir des extensions à HTTP. Les circonstances dans lesquelles la mise en antémémoire d'accréditifs peut interférer avec le modèle de sécurité de l'application incluent sans s'y limiter :

– les clients qui ont été inactifs pendant une longue période à la suite de quoi le serveur peut souhaiter amener le client à réinviter l'utilisateur à présenter ses accréditifs ;

– les applications qui incluent une indication de terminaison de session (comme un bouton 'logout' ou 'commit' sur une page) après quoi le côté serveur de l'application 'sait' qu'il n'y a pas d'autre raison pour que le client conserve les accréditifs.

Ceci fait actuellement l'objet d'une étude distincte. Il y a un certain nombre de travaux autour de divers aspects de ce problème, et on conseille l'utilisation de mots de passe de protection dans les économiseurs d'écran, les temporisations d'inactivité, et autres méthodes qui atténuent les problèmes de sécurité inhérents à cette question. En particulier, les agents d'utilisateur qui mettent en antémémoire les accréditifs sont encouragés à fournir un mécanisme directement accessible pour éliminer les accréditifs en antémémoire sous le contrôle de l'utilisateur.

15.7 Mandataires et mise en antémémoire

Par nature, les mandataires HTTP sont des intermédiaires et représentent une opportunité pour les attaques par intrusion. La compromission des systèmes sur lesquels les mandataires fonctionnent peut déboucher sur de sérieux problèmes de sécurité et de confidentialité. Les mandataires ont accès aux informations qui se rapportent à la sécurité, aux informations personnelles sur les utilisateurs individuels et les organisations, et aux informations particulières qui appartiennent aux utilisateurs et fournisseurs de contenu. Un mandataire compromis, ou un mandataire mis en œuvre ou configuré sans considération des questions de sécurité et de confidentialité pourrait être utilisé dans l'accomplissement d'une vaste gamme d'attaques potentielles. Les opérateurs de mandataires devraient protéger les systèmes sur lesquelles fonctionnent les mandataires comme ils devraient protéger tout système qui contient ou transporte des informations sensibles. En particulier, les informations de journalisation rassemblées auprès des mandataires contiennent souvent des informations personnelles extrêmement sensibles, et/ou des informations sur les organisations. Les informations de journalisation devraient être soigneusement gardées, et des lignes directrices appropriées d'utilisation développées et suivies (paragraphe 15.1.1).

Les mandataires de mise en antémémoire présentent des vulnérabilités potentielles supplémentaires, car le contenu de l'antémémoire représente une cible attractive pour une exploitation malhonnête. Comme le contenu de l'antémémoire persiste après l'achèvement d'une demande HTTP, une attaque de l'antémémoire peut révéler des informations longtemps après qu'un utilisateur pense que ces informations ont été retirées du réseau. Donc, les contenus d'antémémoire devraient être protégés en tant qu'informations sensibles.

Les mises en œuvre de mandataire devraient considérer les implications sur la sécurité et la confidentialité de leurs conceptions et décisions de codage, et des options de configuration qu'ils fournissent aux opérateurs de mandataires (en particulier la configuration par défaut).

Les utilisateurs de mandataires doivent savoir qu'ils ne sont pas plus de confiance que les gens qui font fonctionner le mandataire ; HTTP ne peut par lui-même résoudre ce problème.

L'utilisation judicieuse de la cryptographie, lorsque c'est approprié, peut suffire à protéger contre une large gamme d'attaques contre la sécurité et la confidentialité. Une telle cryptographie sort du domaine d'application de HTTP/1.1.

15.7.1 Attaques de déni de service contre les mandataires

Elles existent. Il est toujours difficile de se défendre contre elles. Les recherches continuent. Attention.

16 Remerciements

La présente spécification fait une grande utilisation du BNF augmenté et des constructions génériques définies par David H. Crocker pour la RFC 822 [9]. De même, elle réutilise beaucoup des définitions fournies par Nathaniel Borenstein et Ned Freed pour MIME [7]. On espère que leur inclusion dans la présente spécification aidera à réduire la confusion passée sur les relations entre les formats de message HTTP et de messagerie Internet.

Le protocole HTTP a considérablement évolué au fil des ans. Il a bénéficié d'une large et active communauté de développeurs – les nombreuses personnes qui ont participé à la liste de diffusion du www-talk -- et c'est cette communauté qui doit le plus être tenue pour responsable du succès de HTTP et de la Toile Mondiale en général. Marc Andreessen, Robert Cailliau, Daniel W. Connolly, Bob Denny, John Franks, Jean-Francois Groff, Phillip M. Hallam-Baker, Hakon W. Lie, Ari Luotonen, Rob McCool, Lou Montulli, Dave Raggett, Tony Sanders, et Marc VanHeyningen méritent une mention spéciale de reconnaissance pour leurs efforts dans la définition des premiers aspects du protocole.

Ce document a largement bénéficié des commentaires de tous les participants au groupe de travail HTTP. En plus de ceux déjà mentionnés, les individus suivants ont contribué à la présente spécification :

Gary Adams	John Klensin	Jeffrey Perry
Harald Tveit Alvestrand	Martijn Koster	Scott Powers
Keith Ball	Alexei Kosut	Owen Rees
Brian Behlendorf	David M. Kristol	Luigi Rizzo
Paul Burchard	Daniel LaLiberte	David Robinson
Maurizio Codogno	Ben Laurie	Marc Salomon
Mike Cowlishaw	Paul J. Leach	Rich Salz
Roman Czyborra	Daniel DuBois	Allan M. Schiffman
Michael A. Dolan	Ross Patterson	Jim Seidman
David J. Fiander	Albert Lunde	Chuck Shotton
Alan Freier	John C. Mallery	Eric W. Sink
Marc Hedlund	Jean-Philippe Martin-Flatin	Simon E. Spero
Greg Herlihy	Mitra	Richard N. Taylor
Koen Holtman	David Morris	Robert S. Thau
Alex Hopmann	Gavin Nicol	Bill (BearHeart) Weinman
Bob Jernigan	David M. Kristol	Francois Yergeau
Shel Kaphan	Bill Perry	Mary Ellen Zurko
Rohit Khare		

Une grande partie du contenu et de la présentation du concept de mise en antémémoire est due aux suggestions et commentaires d'individus parmi lesquels : Shel Kaphan, Paul Leach, Koen Holtman, David Morris, et Larry Masinter.

La plus grande partie de la spécification des gammes est fondée sur un travail fait à l'origine par Ari Luotonen et John Franks, avec des ajouts de Steve Zilles.

Merci au "caviste" de Palo Alto. Il se reconnaîtra.

Jim Gettys (l'éditeur actuel de ce document) souhaite remercier particulièrement Roy Fielding, l'éditeur précédent de ce document, ainsi que John Klensin, Jeff Mogul, Paul Leach, Dave Kristol, Koen Holtman, John Franks, Josh Cohen, Alex Hopmann, Scott Lawrence, et Larry Masinter pour leur aide. Et un merci particulier à Jeff Mogul et Scott Lawrence qui ont effectué la vérification des "DOIT/PEUT/DEVRAIT".

Le groupe Apache, Anselm Baird-Smith, auteur de Jigsaw, et Henrik Frystyk a mis précédemment en œuvre la RFC 2068, et nous les remercions de la découverte de nombre des problèmes que le présent document essaye de rectifier.

17 Références

[1] H. Alvestrand, "Étiquettes pour l'identification des langues"**,** RFC1766, mars 1995. *(Obsolète, voir*

RFC3066, RFC3282) *(P.S.)*

[2] [RFC1436] F. Anklesaria, M. Cahill, P. Lindner, D. Johnson, D. Torrey et B. Albert, "Protocole Gopher Internet (un protocole de recherche et restitution de documents répartis)", mars 1993. *(Information)*

[3]] T. Berners-Lee, "Identifiants de ressource universels dans la Toile mondiale ; syntaxe unificatrice pour l'expression des noms et adresses des objets du réseau utilisés sur la Toile mondiale", RFC1630, juin 1994. *(Information)*

[4] T. Berners-Lee et autres, "Localisateurs uniformes de ressource (URL)", RFC1738, décembre 1994. *(P.S., Obsolète, voir les* RFC4248 *et* 4266*)*

[5] T. Berners-Lee, D. Connolly, "Langage de balisage Hypertext - 2.0", RFC1866, novembre 1995. (*Obsolète, voir* RFC2854) *(Historique)*

[6] T. Berners-Lee, R. Fielding, H. Frystyk, "Protocole de transfert Hypertext -- HTTP/1.0**"**, RFC1945, mai 1996. *(Information)*

[7] N. Freed et N. Borenstein, "Extensions de messagerie Internet multi-objets (MIME) Partie 1 : Format des corps de message Internet", RFC2045, novembre 1996. *(D. S., MàJ par 2184, 2231, 5335.)*

[8] R. Braden, éditeur, "Exigences pour les hôtes Internet – Application et prise en charge", RFC1123, STD 3, octobre 1989.

[9] D. Crocker, "Norme pour le format des messages de texte de l'ARPA-Internet", RFC0822, STD 11, août 1982. (*Obsolète, remplacée par la RFC*5322)

[10] Davis, F., Kahle, B., Morris, H., Salem, J., Shen, T., Wang, R., Sui, J., et M. Grinbaum, "Spécification fonctionnelle du prototype de protocole d'interface WAIS," (v1.5), Thinking Machines Corporation, avril 1990.

[11] R. Fielding, "Localisateurs relatifs de ressource uniforme", RFC1808, juin 1995. (*Obsolète, voir* RFC3986)

[12] M. Horton et R. Adams, "Norme pour l'échange de messages USENET", RFC1036, décembre 1987. *(Remplacée par RFC5536)*

[13] B. Kantor et P. Lapsley, "Protocole de transfert des nouvelles du réseau", RFC0977, février 1986. *(Obsolète, voir RFC3977)*

[14] K. Moore, "MIME (Extensions de messagerie Internet multi-objets) Partie trois : extensions d'en-tête de message pour texte non ASCII", RFC2047, novembre 1996. (*MàJ par* RFC2184, RFC2231) *(D.S.)*

[15] E. Nebel, L. Masinter, "Téléchargement de fichier à base de formulaire dans HTML", RFC1867, novembre 1995. *(Obsolète, voir* RFC2854) *(Historique)*

[16] J. Postel, "Protocole simple de transfert de messagerie", RFC0821, STD 10, août 1982.

[17] J. Postel, "Procédures d'enregistrement des types de support", RFC1590, mars 1994. *(Info., remplacée par les* RFC2045-49)

[18] J. Postel et J. Reynolds, "Protocole de transfert de fichiers (FTP)", RFC0959, STD 9, octobre 1985.

[19] J. Reynolds et J. Postel, "Numéros alloués", RFC1700, STD 2, octobre 1994. *(Historique, voir www.iana.org))*

[20] K. Sollins et L. Masinter, "Exigences fonctionnelles pour les noms de ressource uniformes", RFC1737, décembre 1994.

[21] US-ASCII. Ensemble de caractères codés – Norme américaine de code à 7 bits pour les échanges d'information. Norme ANSI X3.4-1986, ANSI, 1986.

[22] ISO-8859. Norme internationale – Traitement de l'information – Jeux de caractères codés à un seul octet

de 8 bits --
Partie 1 : Alphabet latin n° 1, ISO-8859-1:1987.
Partie 2 : Alphabet latin n° 2, ISO-8859-2, 1987.
Partie 3 : Alphabet latin n° 3, ISO-8859-3, 1988.
Partie 4 : Alphabet latin n° 4, ISO-8859-4, 1988.
Partie 5 : Alphabet latin/cyrillique, ISO-8859-5, 1988.
Partie 6 : Alphabet latin/arabe, ISO-8859-6, 1987.
Partie 7 : Alphabet latin/grec, ISO-8859-7, 1987.
Partie 8 : Alphabet latin/hébreu, ISO-8859-8, 1988.
Partie 9 : Alphabet latin n° 5, ISO-8859-9, 1990.

[23] J. Myers, M. Rose, "Champ d'en-tête Contenu-MD5", RFC1864, octobre 1995. *(D.S.)*

[24] B. Carpenter, Y. Rekhter, "Un dénumérotage représente du travail", RFC1900, février 1996. *(Information)*

[25] P. Deutsch, "Spécification du format de fichier GZIP version 4.3", RFC1952, mai 1996. *(Information)*

[26] Venkata N. Padmanabhan, et Jeffrey C. Mogul. "Amélioration de la latence sur HTTP", Computer Networks et ISDN Systems, v. 28, pp. 25-35, déc 1995. Version légèrement révisée du papier de Proc. 2nd International WWW Conference '94: ʻhébreu et le Web, Oct. 1994, disponible à http://www.ncsa.uiuc.edu/SDG/IT94/Proceedings/DDay/mogul/HTTPLatency.html

[27] Joe Touch, John Heidemann, et Katia Obraczka. "Analyse des performance de HTTP", <URL: http://www.isi.edu/touch/pubs/http-perf96/>, ISI Research Report ISI/RR-98-463, (rapport original daté d'août 1996), USC/Information Sciences Institute, août 1998.

[28] D. Mills, "Protocole de l'heure du réseau, version 3, spécification, mise en œuvre et analyse", RFC1305, STD 12, mars 1992. *(Remplacée par RFC5905)*

[29] P. Deutsch, "Spécification du format DEFLATE de données compressées, version 1.3", RFC1951, mai 1996.

[30] S. Spero, "Analyse des problèmes de performance sur HTTP" http://sunsite.unc.edu/mdma-release/http-prob.html.

[31] P. Deutsch et J-L Gailly, "Spécification du format ZLIB de données compressées, version 3.3", RFC1950, mai 1996.

[32] J. Franks et autres, "Extension à HTTP : authentification d'accès par résumé", RFC2069, janvier 1997. (*Obs., voir* RFC2617) *(P.S.)*

[33] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, T. Berners-Lee, "Protocole de transfert Hypertext -- HTTP/1.1", RFC2068, janvier 1997. (*Obsolète, voir* RFC2616) *(P.S.)*

[34] S. Bradner, "Mots clés à utiliser dans les RFC pour indiquer les niveaux d'exigence", RFC2119, BCP 14, mars 1997.

[35] R. Troost, S. Dorner, "Communication des informations de présentation dans les messages Internet : l'en-tête de disposition de contenu", RFC1806, juin 1995. (*Obsolète, voir* RFC2183) *(Expérimentale)*

[36] J. C. Mogul et autres, "Utilisation et interprétation des numéros de version HTTP", RFC2145, mai 1997. *(Information)*

[37] J. Palme, "En-têtes communs de message Internet", RFC2076, février 1997. *(Information)*

[38] F. Yergeau, "UTF-8, un format de transformation de la norme ISO 10646", RFC2279, janvier 1998. (*Obsolète, voir* RFC3629) *(D.S.)*

[39] Nielsen, H.F., Gettys, J., Baird-Smith, A., Prud'hommeaux, E., Lie, H., et C. Lilley. "Effets de HTTP/1.1 sur les performances du réseau, CSS1, et PNG," Minutes du ACM SIGCOMM '97, Cannes France, septembre 1997.[jg642]

[40] N. Freed et N. Borenstein, "Extensions de messagerie Internet multi-objets (MIME) Partie 2 : Types de support", RFC2046, novembre 1996. *(D. S., MàJ par 2646, 3798, 5147, 6657.)*

[41] H. Alvestrand, "Politique de l'IETF en matière de jeux de caractères et de langages", RFC2277, BCP 18, janvier 1998.

[42] T. Berners-Lee, R. Fielding et L. Masinter, "Identifiants de ressource uniformes (URI) : Syntaxe générique", RFC2396, août 1998. (*Obsolète, voir* RFC3986)

[43] J. Franks et autres, "Authentification HTTP : Authentification d'accès de base et par résumé", RFC2617 , juin 1999. *(DS.)*

[44] Luotonen, A., "Protocoles de tunnelage fondés sur TCP à travers un serveur mandataire de la toile," Travail en cours, (non publié comme RFC). [jg647]

[45] J. Palme, A. Hopmann, "Encapsulation de documents de messagerie électronique MIME agrégés, tels que HTML (MHTML)", RFC2110, mars 1997. (*Obsolète, voir* RFC2557) *(P.S.)*

[46] S. Bradner, "Le processus de normalisation de l'Internet -- Révision 3", RFC2026, (BCP0009) octobre 1996. (*MàJ par* RFC3667, RFC3668, RFC3932, RFC3979, RFC3978, RFC5378, RFC6410)

[47] L. Masinter, "Protocole de contrôle hyper texte de la cafetière (HTCPCP/1.0)", RFC2324, 1er avril 1998. *(Information)*

[48] N. Freed, N. Borenstein, "Extensions multi-objets de la messagerie Internet (MIME) Partie cinq : critères de conformité et exemples", RFC2049, novembre 1996. *(D.S.)*

[49] R. Troost, S. Dorner, K. Moore, éd., "Communication des informations de présentation dans les messages Internet : le champ d'en-tête Contenu-disposition", RFC2183, août 1997. (*MàJ par* RFC2184, RFC2231) *(P.S.)*

18 Adresse des auteurs

Roy T. Fielding
Information et Computer Science
University of California, Irvine
Irvine, CA 92697-3425,
USA
Fax : +1 (949) 824-1715
mél : fielding@ics.uci.edu

James Gettys
World Wide Web Consortium
MIT Laboratory for Computer Science
545 Technology Square
Cambridge, MA 02139, USA
Fax : +1 (617) 258 8682
mél : jg@w3.org

Jeffrey C. Mogul
Western Research Laboratory
Compaq Computer Corporation
250 University Avenue
Palo Alto, California, 94305, USA
mél : mogul@wrl.dec.com

Larry Masinter
Xerox Corporation
3333 Coyote Hill Road
Palo Alto, CA 94034,
USA

mél : masinter@parc.xerox.com

Tim Berners-Lee
Director, World Wide Web Consortium
MIT Laboratory for Computer Science
545 Technology Square
Cambridge, MA 02139, USA
Fax : +1 (617) 258 8682
mél : timbl@w3.org

Paul J. Leach
Microsoft Corporation
1 Microsoft Way
Redmond, WA 98052,
USA

mél : paulle@microsoft.com

Henrik Frystyk Nielsen
World Wide Web Consortium
MIT Laboratory for Computer Science
545 Technology Square
Cambridge, MA 02139, USA
Fax : +1 (617) 258 8682
mél : frystyk@w3.org

19 Appendices

19.1 Type de support Internet message/http et application/http

En plus de la définition du protocole HTTP/1.1, le présent document sert de spécification pour le type de support Internet "message/http" et "application/http". Le type message/http peut être utilisé pour inclure une seule demande HTTP ou message de réponse, pourvu qu'elle obéisse aux restrictions de MIME pour tous les types de "message" concernant la longueur de ligne et les codages. Le type application/http peut être utilisé pour inclure un traitement en parallèle *(pipeline)* d'une ou plusieurs demandes ou messages de réponse HTTP (non imbriqués). Ce qui suit est à enregistrer auprès de IANA [17].

Nom de type de support	message
Nom de sous-type de support	http
Paramètres exigés	aucun
Paramètres facultatifs	version, msgtype
Version	Le numéro HTTP-Version du message inclus (par exemple, "1.1"). Si il n'est pas présent, la version peut être déterminée à partir de la première ligne du corps.
msgtype	Le type de message -- "request" ou "response". Si il n'est pas présent, le type peut être déterminé à partir de la première ligne du corps.
Considérations de codage	seuls "7bit", "8bit", ou "binary" sont permis
Considérations de sécurité	aucune

Nom de type de support	application
Nom de sous-type de support	http
Paramètres exigés	aucun
Paramètres facultatifs	version, msgtype
Version	Le numéro HTTP-Version du message inclus (par exemple, "1.1"). Si il n'est pas présent, la version peut être déterminée à partir de la première ligne du corps.
msgtype	Le message type -- "request" ou "response". Si il n'est pas présent, le type peut être déterminé à partir de la première ligne du corps.
Considérations de codage	Les messages HTTP inclus par ce type sont en format "binary" ; l'utilisation d'un Content-Transfer-Encoding approprié est exigé lors d'une transmission par messagerie électronique.
Considérations de sécurité	aucune

19.2 Type de support Internet multipart/byteranges

Lorsqu'un message de réponse HTTP 206 (Contenu partiel) inclut le contenu de plusieurs gammes (une réponse à une demande pour plusieurs gammes qui ne se recouvrent pas) celles-ci sont transmises comme un corps de message multiparties. Le type de support pour cela est appelé "multipart/byteranges".

Le type de support multipart/byteranges inclut deux parties ou plus, chacune ayant ses propres champs Content-Type et Content-Range. Le paramètre exigé boundary spécifie la chaîne de limite utilisée pour séparer chaque partie de corps.

Nom de type de support	multipart
Nom de sous-type de support	byteranges
Paramètres exigés	boundary
Paramètres facultatifs	aucune
Considérations de codage	seuls les "7bit", "8bit", ou "binary" sont permis
Considérations de sécurité	aucune

Par exemple :

```
HTTP/1.1 206 Contenu partiel
Date: Wed, 15 Nov 1995 06:25:24 GMT
Last-Modified: Wed, 15 Nov 1995 04:58:08 GMT
Content-type: multipart/byteranges; boundary=THIS_STRING_SEPARATES
```

```
--THIS_STRING_SEPARATES
Content-type: application/pdf
Content-range: bytes 500-999/8000

...la première gamme...
--THIS_STRING_SEPARATES
Content-type: application/pdf
Content-range: bytes 7000-7999/8000

...la seconde gamme
--THIS_STRING_SEPARATES--
```

Notes:
1) Des CRLF additionnels peuvent précéder la première chaîne de limite dans l'entité.

2) Bien que la RFC 2046 [40] permette que la chaîne de limite soit entre guillemets, certaines mises en œuvre existantes traitent une chaîne de limite entre guillemets comme incorrecte.

3) Un certain nombre de navigateurs et serveurs ont été codés sur un projet précoce de spécification de gamme d'octets pour utiliser un type de support de multipart/x-byteranges, qui est presque, mais pas tout à fait, compatible avec la version exposée dans HTTP/1.1.

19.3 Applications tolérantes

Bien que le présent document spécifie les exigences pour la génération des messages HTTP/1.1, toutes les applications ne les mettront pas correctement en œuvre. On recommande donc que les applications opérationnelles soient tolérantes pour les variantes, chaque fois que ces variantes peuvent être interprétées sans ambiguïté.

Les clients, dans l'analyse de la ligne d'état, et les serveurs lors de l'analyse de la ligne de demande, DEVRAIENT être tolérants. En particulier, ils DEVRAIENT accepter toute quantité de caractères SP ou HT entre les champs, quand bien même un seul SP est exigé.

La terminaison de ligne pour les champs d'en-tête de message est le CRLF de séquence. Cependant, on recommande que les applications, lors de l'analyse de tels en-têtes, reconnaissent un seul LF comme terminaison de ligne et ignorent le CR qui se trouve devant.

Le jeu de caractères d'un corps d'entité DEVRAIT être étiqueté comme le plus petit commun dénominateur des codes de caractères utilisés dans ce corps, avec l'exception que ne pas étiqueter est préféré à étiqueter l'entité avec les étiquettes US-ASCII ou ISO-8859-1. Voir aux paragraphes 3.7.1 et 3.4.1.

Des règles supplémentaires pour les exigences sur l'analyse et le codage des dates et autres problèmes potentiels avec les codages de date sont :

- Les clients et antémémoires HTTP/1.1 DEVRAIENT supposer qu'une date de la RFC-850 qui apparaît être à plus de 50 ans dans le futur est en fait dans le passé (ce qui aide à résoudre le problème de "l'an 2000").
- Une mise en œuvre HTTP/1.1 PEUT représenter en interne une analyse de date Expires comme plus tôt que la valeur correcte, mais NE DOIT PAS représenter en interne une analyse de date Expires comme plus tard que la valeur correcte.
- Tous les calculs en rapport avec l'expiration DOIVENT être effectués en GMT. La zone d'heure locale NE DOIT PAS influencer le calcul ou la comparaison d'un âge ou d'une heure d'expiration.
- Si un en-tête HTTP porte incorrectement une valeur de date avec une zone horaire autre que GMT, il DOIT être converti en GMT en utilisant la conversion la plus prudente possible.

19.4 Différences entre entités HTTP et entités de la RFC 2045

HTTP/1.1 utilise de nombreuses constructions définies pour la messagerie Internet (RFC 822 [9]) et les

extensions multi-usage de messagerie Internet (MIME [7]) pour permettre aux entités d'être transmises dans des représentations diverses et par des mécanismes extensibles. Cependant, la RFC 2045 discute de messagerie, et HTTP a quelques caractéristiques qui sont différentes de celles décrites dans la RFC 2045. Ces différences ont été soigneusement choisies pour optimiser les performances sur les connexions binaires, pour permettre une plus grande liberté dans l'utilisation de nouveaux types de support, pour rendre les comparaisons plus faciles, et pour tenir compte des pratiques des premiers serveurs et clients HTTP.

Le présent appendice décrit les zones spécifiques où HTTP diffère de la RFC 2045. Les mandataires et passerelles vers des environnements MIME stricts DEVRAIENT être avisés de ces différences et fournir les conversions appropriées lorsque nécessaire. Les mandataires et passerelles d'environnements MIME vers HTTP doivent aussi être avisés de ces différences parce que certaines conversions pourraient être exigées.

19.4.1 MIME-Version

HTTP n'est pas un protocole conforme à MIME. Cependant, les messages HTTP/1.1 PEUVENT inclure un seul champ d'en-tête général MIME-Version pour indiquer quelle version du protocole MIME a été utilisée pour construire le message. L'utilisation du champ d'en-tête MIME-Version indique que le message est pleinement conforme au protocole MIME (comme défini à la RFC 2045 [7]). Les mandataires/passerelles sont chargés d'assurer la pleine conformité (si possible) lors de l'exportation des messages HTTP vers des environnements MIME stricts.

```
MIME-Version   = "MIME-Version" ":" 1*DIGIT "." 1*DIGIT
```

MIME version "1.0" est l'utilisation par défaut pour HTTP/1.1. Cependant, l'analyse et la sémantique de message HTTP/1.1 sont définies par le présent document et non par la spécification MIME.

19.4.2 Conversion en forme canonique

La RFC 2045 [7] exige qu'une entité de messagerie Internet soit convertie en forme canonique avant d'être transférée, comme décrit à la Section 4 de la RFC 2049 [48]. Le paragraphe 3.7.1 du présent document décrit les formes permises pour les sous-types du support "text" lorsqu'ils sont transmis sur HTTP. La RFC 2046 exige que le contenu du type "text" représente les coupures de ligne comme des CRLF et interdit l'utilisation de CR ou LF en dehors des séquences de rupture de ligne. HTTP permet le CRLF, interdit le CR, et interdit le LF pour indiquer une rupture de ligne dans le contenu du texte lorsqu'un message est transmis sur HTTP.

Lorsque c'est possible, un mandataire ou passerelle de HTTP vers un environnement MIME strict DEVRAIT traduire toutes les coupures de ligne dans les types de support text décrits au paragraphe 3.7.1 du présent document en forme canonique de CRLF de la RFC 2049. Noter cependant, que ceci peut être compliqué par la présence d'un codage de contenu et par le fait que HTTP permet l'utilisation de certains jeux de caractères qui n'utilisent pas les octets 13 et 10 pour représenter CR et LF, comme c'est le cas pour certains jeux de caractères multi-octets.

Les mises en œuvre devraient noter que la conversion cassera toute somme de contrôle cryptographique appliquée au contenu original sauf si le contenu original est déjà sous forme canonique. Donc, la forme canonique est recommandée pour tout contenu qui utilise de telles sommes de contrôle dans HTTP.

19.4.3 Conversion des formats de date

HTTP/1.1 utilise un ensemble restreint de formats de date (paragraphe 3.3.1) pour simplifier le traitement des comparaisons de date. Les mandataires et passerelles provenant d'autres protocoles DEVRAIENT s'assurer que tout champ d'en-tête Date présent dans un message se conforme à un des formats HTTP/1.1 et réécrire la date si nécessaire.

19.4.4 Introduction de Content-Encoding

La RFC 2045 n'inclut aucun concept équivalent au champ d'en-tête Content-Encoding de HTTP/1.1. Comme celui-ci agit comme modificateur du type de support, les mandataires et passerelles de HTTP vers des protocoles conformes à MIME DOIVENT soit changer la valeur du champ d'en-tête Content-Type, soit décoder le corps d'entité, avant de transmettre le message. (Certaines applications expérimentales de Content-Type

pour la messagerie Internet ont utilisé un paramètre de type de support de ";conversions=<content-coding>" pour effectuer une fonction équivalente à Content-Encoding. Cependant, ce paramètre ne fait pas partie de la RFC 2045.)

19.4.5 Pas de Content-Transfer-Encoding

HTTP n'utilise pas le champ Content-Transfer-Encoding (CTE) de la RFC 2045. Les mandataires et passerelles de protocoles conformes à MIME vers HTTP DOIVENT retirer tout codage CTE qui n'est pas une identité ("quoted-printable" ou "base64") avant de livrer le message de réponse à un client HTTP.

Les mandataires et passerelles de HTTP vers des protocoles conformes à MIME sont chargés de s'assurer que le message est dans le format et codage correct pour un transport sûr sur ce protocole, où "transport sûr" est défini par les limitations au protocole à utiliser. Un tel mandataire ou passerelle DEVRAIT étiqueter les données avec un Content-Transfer-Encoding approprié si le faisant, cela améliore la probabilité d'un transport sûr sur le protocole de destination.

19.4.6 Introduction of Transfer-Encoding

HTTP/1.1 introduit le champ d'en-tête Transfer-Encoding (paragraphe 14.41). Les mandataires et passerelles DOIVENT retirer tout codage de transfert avant de transmettre un message via un protocole conforme à MIME.

Un processus de décodage de codage de transfert "fragmenté" (paragraphe 3.6) peut être représenté en pseudo-code par :

```
longueur := 0
  lire taille-de-fragment, extension-de-fragment (s'il en est) et CRLF
  si (taille-de-fragment > 0) {
  lire données-de-fragment et CRLF
  ajouter les données du fragment au corps d'entité
  longueur := longueur + taille-de-fragment
  lire taille-de-fragment et CRLF
  }
  lire en-tête-d'entité
  si (en-tête-d'entité pas vide) {
  ajouter en-tête-d'entité aux champs d'en-tête existants
  lire en-tête-d'entité
  }
  Content-Length := longueur
  Retirer "fragmenté" de Transfer-Encoding
```

19.4.7 MHTML et limitation de longueur de ligne

Les mises en œuvre HTTP qui partagent leur code avec les mises en œuvre MHTML [45] doivent être au courant des limitations de longueur de ligne de MIME. Comme HTTP n'a pas cette limitation, HTTP ne coupe pas les lignes longues. Les messages MHTML transportés par HTTP suivent toutes les conventions de MHTML, y compris les limitations de longueur et coupures de ligne, la canonisation, etc., car HTTP transporte tous les corps de message comme des charges utiles (voir au paragraphe 3.7.2) et n'interprète pas le contenu ou aucune des lignes d'en-tête MIME qui pourraient y être contenues.

19.5 Caractéristiques supplémentaires

Les RFC 1945 et RFC 2068 décrivent les éléments de protocole utilisés par certaines des mises en œuvre HTTP existantes, mais pas de façon cohérente et correcte à travers la plupart des applications HTTP/1.1. Il est conseillé aux mises en œuvre d'être au courant de ces caractéristiques, mais elles ne peuvent pas s'appuyer sur leur présence, ou de leur interopérabilité, avec d'autres applications HTTP/1.1. Certaines d'entre elles décrivent des caractéristiques expérimentales proposées, et certaines décrivent des caractéristiques dont les développements expérimentaux ont montré le manque et qui sont maintenant traités dans la spécification HTTP/1.1 de base.

Un certain nombre d'autres en-têtes, comme Content-Disposition et Title, provenant de SMTP et MIME sont aussi souvent mis en œuvre (voir la RFC 2076 [37]).

19.5.1 Content-Disposition

Le champ d'en-tête de réponse Content-Disposition a été proposé comme un moyen pour que le serveur d'origine suggère un nom de fichier par défaut si l'utilisateur demande que le contenu soit sauvegardé dans un fichier. Cet usage est dérivé de la définition de Content-Disposition dans la RFC 1806 [35].

```
content-disposition = "Content-Disposition" ":"
disposition-type *( ";" disposition-parm )
disposition-type = "attachment" | disp-extension-token
disposition-parm = filename-parm | disp-extension-parm
filename-parm = "filename" "=" quoted-string
disp-extension-token = token
disp-extension-parm = token "=" ( token | quoted-string )
```

Par exemple

```
Content-Disposition: attachment; filename="fname.ext"
```

L'agent d'utilisateur receveur NE DEVRAIT PAS respecter les informations de chemin de répertoire présentes dans le paramètre filename-parm, qui est le seul paramètre connu pour s'appliquer aux mises en œuvre HTTP à l'heure actuelle. Le filename DEVRAIT être traité seulement comme un composant terminal.

Si cet en-tête est utilisé dans une réponse avec le type de contenu application/octet-stream, la suggestion implicite est que l'agent d'utilisateur ne devrait pas afficher la réponse, mais entrer directement un dialogue `save response as...'.

Voir au paragraphe 15.5 les questions de sécurité pour Content-Disposition.

19.6 Compatibilité avec les versions précédentes

Il sort du domaine d'application d'une spécification de protocole de rendre obligatoire la conformité aux versions précédentes. HTTP/1.1 a été cependant délibérément conçu pour faciliter la prise en charge des versions précédentes. Il vaut de noter que, au moment de la composition de la présente spécification (1996), on s'attend à ce que les serveurs de HTTP/1.1 commercial :

- reconnaissent le format de la ligne de demande pour des demandes HTTP/0.9, 1.0, et 1.1 ;
- comprennent toute demande valide dans les formats HTTP/0.9, 1.0, ou 1.1 ;
- répondent de façon appropriée par un message dans la même version majeure qu'utilisée par le client.

Et on s'attend à ce que les clients HTTP/1.1 :

- reconnaissent le format de la ligne d'état pour les réponses HTTP/1.0 et 1.1 ;
- comprennent toute réponse valide dans les formats HTTP/0.9, 1.0, ou 1.1.

Pour la plupart des mises en œuvre de HTTP/1.0, chaque connexion est établie par le client avant la demande et close par le serveur après l'envoi de la réponse. Certaines mises en œuvre utilisent la version Keep-Alive des connexions persistantes décrites au paragraphe 19.7.1 de la RFC 2068 [33].

19.6.1 Changements par rapport à HTTP/1.0

Ce paragraphe résume les différences majeures entre les versions HTTP/1.0 et HTTP/1.1.

19.6.1.1 Changements pour simplifier les serveurs Web multi-domiciliés et conserver les adresses IP

Les exigences que les clients et serveurs prennent en charge l'en-tête de demande Host, rapportent une erreur si l'en-tête de demande Host (paragraphe 14.23) manque dans une demande HTTP/1.1, et acceptent les URI

absolus (paragraphe 5.1.2) sont les changements les plus importants définis par cette spécification.

Les clients HTTP/1.0 plus anciens supposent une relation univoque des adresses et des serveurs ; il n'y avait pas d'autre mécanisme établi pour distinguer le serveur voulu pour une demande que l'adresse IP à laquelle cette demande était dirigée. Le changement mentionné ci-dessus permet à l'Internet, une fois que les clients HTTP plus anciens ne seront plus très courants, de prendre en charge des sites Web multiples à partir une seule adresse IP, simplifiant considérablement les grands serveurs Web opérationnels, où l'allocation de plusieurs adresses IP à un seul hôte a créé de sérieux problèmes. L'Internet sera aussi capable de récupérer les adresses IP qui ont été allouées dans le seul but de permettre d'utiliser des noms de domaine à but particulier dans des URL HTTP de niveau racine. Étant donné le taux de croissance de la Toile, et le nombre de serveurs déjà déployés, il est extrêmement important que toutes les mises en œuvre de HTTP (y compris les mises à jour des applications HTTP/1.0 existantes) mettent en œuvre correctement ces exigences :

- Clients et serveurs DOIVENT tous deux prendre en charge l'en-tête de demande Host.
- Un client qui envoie une demande HTTP/1.1 DOIT envoyer un en-tête Host.
- Les serveurs DOIVENT rapporter une erreur 400 (Mauvaise demande) si une demande HTTP/1.1 ne comporte pas d'en-tête de demande Host.
- Les serveurs DOIVENT accepter les URI absolus.

19.6.2 Compatibilité avec les connexions persistantes HTTP/1.0

Certains clients et serveurs peuvent souhaiter être compatibles avec certaines mises en œuvre précédentes de connexions persistantes dans des clients et serveurs HTTP/1.0. Les connexions persistantes dans HTTP/1.0 sont négociées explicitement car elles ne sont pas le comportement par défaut. Les mises en œuvre HTTP/1.0 expérimentales de connexions persistantes sont déficientes et les nouvelles fonctionnalités de HTTP/1.1 sont conçues pour rectifier ces problèmes. Le problème était que certains clients 1.0 existants peuvent envoyer Keep-Alive à un serveur mandataire qui ne comprend pas Connection, et qui le transmettrait fautivement au prochain serveur entrant, lequel établirait la connexion Keep-Alive et il en résulterait un mandataire HTTP/1.0 faisant le pied de grue en attendant la clôture suite à la réponse. Le résultat est que les clients HTTP/1.0 doivent être empêchés d'utiliser Keep-Alive lors d'un dialogue avec un mandataire.

Cependant, le dialogue avec les mandataires est l'utilisation la plus importante des connexions persistantes, de sorte que cette prohibition est inacceptable. Nous avons donc besoin d'un autre mécanisme pour indiquer qu'une connexion persistante est souhaitée, qui soit d'utilisation sûre même lors d'un dialogue avec un vieux mandataire qui ignore Connection. Les connexions persistantes sont le comportement par défaut pour les messages HTTP/1.1 ; on introduit un nouveau mot clé (Connection: close) pour déclarer la non-persistance. Voir au paragraphe 14.10.

La forme HTTP/1.0 originale des connexions persistantes (Connection: Keep-Alive et l'en-tête Keep-Alive) est exposée dans la RFC 2068. [33]

19.6.3 Changements par rapport à la RFC 2068

La présente spécification a été inspectée avec le plus grand soin pour corriger et résoudre les ambiguïtés de l'utilisation des mots clés ; la RFC 2068 posait de nombreux problèmes par rapport aux conventions établies dans la RFC 2119 [34].

Il a été précisé quel code d'erreur devrait être utilisé pour les défaillances de serveur entrant (par exemple défaillances de DNS). (Paragraphe 10.5.5).

CREATE avait une compétition qui exigeait qu'une ETag soit envoyée lorsqu'une ressource était créée. (Paragraphe 10.2.2).

Content-Base a été supprimé de la spécification : il n'était pas largement mis en œuvre, et il n'y a pas de façon simple et sûre de l'introduire sans un robuste mécanisme d'extension. De plus, il est utilisé d'une façon similaire, mais pas de façon identique dans MHTML [45].

Le codage de transfert et les longueurs de message interagissent tous d'une façon qui exige de fixer exactement quand le codage fragmenté est utilisé (pour permettre que le codage de transfert ne puisse pas s'auto délimiter) ; il était important de préciser exactement comment sont calculées les longueurs de message

(Paragraphes 3.6, 4.4, 7.2.2, 13.5.2, 14.13, 14.16).

Un codage de contenu de "identity" a été introduit, pour résoudre des problèmes découverts dans la mise en antémémoire (paragraphe 3.5).

Les valeurs de qualité de zéro devraient indiquer que "Je ne veux pas quelque chose" pour permettre aux clients de refuser une représentation (paragraphe 3.9).

L'utilisation et l'interprétation des numéros de version HTTP ont été précisées par la RFC 2145. Elle exige des mandataires qu'ils mettent à niveau les demandes sur la plus haute version de protocole qu'ils prennent en charge pour régler les problèmes découverts dans les mises en œuvre HTTP/1.0 (paragraphe 3.1).

La mise en caractères génériques des jeux de caractères est introduite pour éviter l'explosion des noms de jeu de caractères dans les en-têtes accept (paragraphe 14.2).

Un cas manquait dans le modèle Cache-Control de HTTP/1.1 ; s-maxage a été introduit pour ajouter ce cas manquant (paragraphes 13.4, 14.8, 14.9, 14.9.3).

La directive Cache-Control: max-âge n'était pas définie correctement pour les réponses (paragraphe 14.9.3).

Il y a des situations où un serveur (en particulier un mandataire) ne connaît pas la longueur totale d'une réponse mais est capable de servir une demande byterange. On a donc besoin d'un mécanisme pour admettre byterange avec une gamme de contenu n'indiquant pas la longueur totale du message (paragraphe 14.16).

Les réponses de demande de gamme vont devenir très fournies si toutes les méta-données sont toujours retournées ; en permettant au serveur d'envoyer seulement les en-têtes nécessaires dans une réponse 206, ce problème peut être évité (paragraphes 10.2.7, 13.5.3, et 14.27).

Régler le problème des demandes de gamme qu'on ne peut pas satisfaire ; il y a deux cas : les problèmes de syntaxe, et il n'existe pas de gamme dans le document. Le code d'état 416 était nécessaire pour résoudre cette ambiguïté afin d'indiquer une erreur pour une demande de gamme d'octets qui tombe en dehors du contenu réel d'un document (paragraphe 10.4.17, 14.16).

Réécrire les exigences de transmission de message pour rendre plus difficile aux mises en oeuvre de le faire mal, car les conséquences des erreurs ont ici un impact significatif sur l'Internet, et pour traiter les problèmes suivants:

1. Changer "HTTP/1.1 ou plus récent" en "en "HTTP/1.1", dans les contextes où cela plaçait à tort une exigence sur le comportement d'une mise en œuvre d'une future version de HTTP/1.x
2. Préciser que les agents d'utilisateur devraient réessayer les demandes, et non les "clients" en général.
3. Convertir les exigences que les clients ignorent les réponses 100 (Continue) inattendues, et pour les mandataires de transmettre les réponses 100, en une exigence générale de réponses 1xx.
4. Modifier certains langages spécifiques de TCP, pour préciser que les transports non-TCP sont possibles pour HTTP.
5. Exiger que le serveur d'origine NE DOIT PAS attendre le corps de la demande avant qu'il envoie une réponse 100 (Continue) exigée.
6. Permettre, plutôt qu'exiger, qu'un serveur omette 100 (Continue) si il a déjà vu une partie du corps de la demande.
7. Permettre aux serveurs de se défendre contre les attaques de déni de service et les clients en panne.

Ces changements ajoutent l'en-tête Expect et le code d'état 417. Les exigences de transmission de message sont aux paragraphes 8.2, 10.4.18, 8.1.2.2, 13.11, et 14.20.

Les mandataires devraient être capables d'ajouter Content-Length lorsque c'est approprié (paragraphe 13.5.2).

Suppression de la confusion entre réponses 403 et 404 (paragraphes 10.4.4, 10.4.5, et 10.4.11).

Les avertissements pourraient être mis incorrectement en antémémoire, ou mis à jour de façon non appropriée. (Paragraphes 13.1.2, 13.2.4, 13.5.2, 13.5.3, 14.9.3, et 14.46) Warning doit aussi être un en-tête général, car

PUT ou d'autres méthodes peuvent avoir besoin de lui dans des demandes.

Le codage de transfert pose des problèmes significatifs, en particulier dans les interactions avec le codage fragmenté. La solution est que les codages de transfert deviennent de plein droit comme les codages de contenu. Ceci implique l'ajout d'un registre IANA pour les codages de transfert (séparé des codages de contenu) un nouveau champ d'en-tête (TE) et l'activation à l'avenir d'en-têtes de queue. Le codage de transfert est un avantage majeur pour les performances, et donc il vaut la peine de le régler [39]. TE aussi résout un autre problème, obscur, d'interopérabilité vers l'aval qui pourrait survenir du fait des interactions entre les en-queue d'authentification, le codage fragmenté et les clients HTTP/1.0 (paragraphes 3.6, 3.6.1, et 14.39).

Les méthodes PATCH, LINK, UNLINK étaient définies mais n'étaient pas communément mises en œuvre dans les précédentes versions de cette spécification. Voir la RFC 2068 [33].

Les champs d'en-tête Alternates, Content-Version, Derived-From, Link, URI, Public et Content-Base étaient définis dans les versions précédentes de cette spécification, mais pas communément mises en oeuvre. Voir la RFC 2068 [33].

20 Index

Prière de se reporter à la version PostScript de la présente RFC.

21 Déclaration de droits de reproduction