Groupe de travail Réseau
Demande for Comments : 4918
RFC rendue obsolète : 2518 ; mise à jour par RFC 5689
Catégorie : Sur la voie de la normalisation

L. Dusseault, éditrice
CommerceNet
juin 2007
Traduction Claude Brière de L'Isle

Extensions HTTP pour les informations d'auteur et de version distribuées sur la Toile (WebDAV)

Statut du présent mémoire
Le présent document spécifie un protocole sur la voie de la normalisation de l'Internet pour la communauté de l'Internet, et appelle à des discussions et suggestions pour son amélioration. Prière de se référer à l'édition en cours des "Normes officielles des protocoles de l'Internet" (STD 1) pour connaître l'état de la normalisation et le statut de ce protocole. La distribution du présent mémoire n'est soumise à aucune restriction.
(La présente traduction incorpore les errata 1068, 1207, 1371, 1430, 1519, 1635, 4435)

Résumé
Les informations d'auteur et de version distribuées sur la Toile (WebDAV, *Web Distributed Authoring et Versioning*) consistent en un ensemble de méthodes, en-têtes, et types de contenu auxiliaires de HTTP/1.1 pour la gestion des propriétés de ressources, la création et la gestion de collections de ressources, la manipulation d'espace de noms d'URL, et le verrouillage de ressources (évitement de collision).

La RFC 2518 a été publiée en février 1999, et la présente spécification la rend obsolète avec des révisions mineures dues principalement à l'expérience de l'interopérabilité.

Table des Matières

1. Introduction

Le présent document décrit une extension au protocole HTTP/1.1 qui permet aux clients d'effectuer à distance des opérations d'auteur de contenu de la Toile. Cette extension fournit un ensemble cohérent de méthodes, en-têtes, formats de corps d'entité de demande, et formats de corps d'entité de réponse qui fournissent des opérations pour des :

Propriétés : la capacité de créer, supprimer, et interroger des informations sur des pages de la Toile, comme leurs auteurs, leur date de création, etc.

Collections : la capacité de créer des ensembles de documents et de restituer une liste hiérarchique des membres (comme un répertoire dans un système de fichiers).

Verrouillage : la capacité d'empêcher plus d'une personne de travailler en même temps sur un document. Cela empêche le "problème de la mise à jour perdue", dans lequel les modifications sont perdues lorsque un auteur, puis un autre, écrit des changements sans les fusionner avec les changements de l'autre auteur.

Opérations d'espace de noms : la capacité de donner pour instruction au serveur de copier et déplacer des ressources de la Toile, opérations qui changent la transposition des URL en ressources.

Les exigences et les raisons de ces opérations sont décrites dans un document d'accompagnement, "Exigences pour un protocole réparti des noms d'auteurs et des numéros de version pour la Toile mondiale" [RFC2291].

Le présent document ne spécifie pas les opérations sur les versions suggérées dans la [RFC2291]. Ce travail a été fait dans un autre document, "Extensions de versions à WebDAV" [RFC3253].

Les Sections qui suivent font une introduction détaillée aux diverses abstractions WebDAV, spécifiquement : propriétés des ressources (Section 4), collections de ressources (Section 5), verrouillages (Section 6) en général, et verrouillages en écriture (Section 7).

Ces abstractions sont manipulées par les méthodes HTTP spécifiques de WebDAV (Section 9) et les en-têtes HTTP supplémentaires (Section 10) utilisés avec les méthodes WebDAV. Les considérations générales pour le traitement des demandes et réponses HTTP dans WebDAV sont à la Section 8.

Bien que les codes d’état fournis par HTTP/1.1 soient suffisants pour décrire la plupart des conditions d'erreur rencontrées par les méthodes WebDAV, il y a quelques erreurs qui ne rentrent pas nettement dans les catégories existantes. La présente spécification définit des codes d’état supplémentaires développés pour les méthodes WebDAV (Section 11) et décrit les codes d’état HTTP existants (Section 12) tels qu'utilisés dans WebDAV. Comme certaines méthodes WebDAV peuvent opérer sur de nombreuses ressources, la réponse multi-états (Section 13) a été introduite pour retourner les informations d’état pour plusieurs ressources. Finalement, cette version de WebDAV introduit des éléments XML de précondition et de postcondition (Section 16) dans les corps de réponse d’erreur.

WebDAV utilise XML [XML] pour les noms et certaines valeurs de propriétés, et utilise aussi XML pour surveiller les demandes et réponses compliquées. La présente spécification contient des DTD et les définitions textuelles de toutes les propriétés (Section 15) et de tous les autres éléments XML (Section 14) utilisés dans la surveillance. WebDAV inclut quelques règles spéciales sur l'extension de la surveillance WebDAV XML de façon rétro compatible (Section 17).

Pour finir la spécification, des sections décrivent ce que signifie pour une ressource d'être conforme à la présente spécification (Section 18), la prise en charge de l'internationalisation (Section 19), et la sécurité (Section 20).

2. Conventions de notation

Comme le présent document décrit un ensemble d'extensions au protocole HTTP/1.1, le BNF augmenté utilisé ici pour décrire les éléments de protocole est exactement le même que celui décrit au paragraphe 2.1 de la [RFC2616], incluant les règles sur les espaces blanches linéaires implicites. Comme ce BNF augmenté utilise les règles de base de production fournies au paragraphe 2.2 de la [RFC2616], ces règles s'appliquent aussi au présent document. Noter que ce n'est pas la syntaxe de BNF standard utilisée dans les autres RFC.

Les mots clés "DOIT", "NE DOIT PAS", "EXIGE", "DEVRA", "NE DEVRA PAS", "DEVRAIT", "NE DEVRAIT PAS", "RECOMMANDE", "PEUT", et "FACULTATIF" dans le présent document sont à interpréter comme décrit dans la [RFC2119].

Noter qu'en langage naturel, une propriété comme "creationdate dans l'espace de noms XML de DAV:" est parfois abrégée en "DAV:creationdate".

3. Terminologie

URI/URL - respectivement, un identifiant de ressource universel et un localisateur de ressource universel. Ces termes (et la distinction entre eux) sont définis dans la [RFC3986].

Transposition URI/URL - relation entre un URI absolu et une ressource. Comme une ressource peut représenter des éléments qui ne sont pas restituables par le réseau, ainsi que d'autres qui le sont, il est possible qu'une ressource ait zéro, une, ou plusieurs transpositions d'URI. La transposition d'une ressource en un URI de schéma "http" rend possible de soumettre des demandes de protocole HTTP à la ressource en utilisant l'URI.

Segment de chemin - informellement, les caractères qui se trouvent entre des barres obliques ("/") dans un URI. Formellement, comme défini au paragraphe 3.3 de la [RFC3986].

Collection - informellement, une ressource qui agit aussi comme un conteneur de références à des fichiers de ressources. Formellement, une ressource qui contient un ensemble de transpositions entre des segments de chemin et des ressources et satisfait aux exigences définies à la Section 5.

Membre interne (d'une collection) - informellement, une ressource fille d'une collection. Formellement, une ressource référencée par une transposition de segment de chemin contenue dans la collection.

URL de membre interne (d'une collection) - URL d'un membre interne, consistant en l'URL de la collection (incluant la barre oblique de fin) plus le segment de chemin qui identifie le membre interne.

Membre (d'une collection) - informellement, un "descendant" d'une collection. Formellement, un membre interne de la collection, ou, de façon récurrente, un membre d'un membre interne.

URL membre (d'une collection) - un URL c'est-à-dire, soit un URL de membre interne de la collection elle-même, soit un URL de membre interne d'un membre de cette collection.

Propriété - paire de nom/valeur qui contient des informations décrivant une ressource.

Propriété vive - propriété dont la sémantique et la syntaxe sont appliquées par le serveur. Par exemple, la propriété vive DAV:getcontentlength a sa valeur, la longueur de l'entité retournée par une demande GET, automatiquement calculée par le serveur.

Propriété morte - propriété dont la sémantique et la syntaxe ne sont pas appliquées par le serveur. Le serveur enregistre seulement la valeur d'une propriété morte ; le client est chargé de maintenir la cohérence de la syntaxe et de la sémantique d'une propriété morte.

Principal - être humain distinct ou acteur de calcul qui initie l'accès à des ressources du réseau.

Jeton d'état *(State Token)* - URI qui représente un état d'une ressource. Les jetons de verrouillage sont les seuls jetons d'état définis dans la présente spécification.

4. Modèle de données pour les propriétés de ressource

4.1 Modèle de propriété de ressource

Les propriétés sont des éléments de données qui décrivent l'état d'une ressource. Les propriétés sont des données sur des données.

Les propriétés sont utilisées dans les environnements d'auteurs répartis pour assurer une découverte et une gestion efficace des ressources. Par exemple, une propriété 'subject' pourrait permettre l'indexation de toutes les ressources par leur sujet, et une propriété 'author' pourrait permettre la découverte de quels auteurs ont écrit quels documents.

Le modèle de propriété DAV consiste en paires de nom/valeur. Le nom d'une propriété identifie la syntaxe et la sémantique

de la propriété, et fournit une adresse par laquelle se référer à sa syntaxe et sa sémantique.

Il y a deux catégories de propriétés : "live" *(vive)* et "dead" *(morte)*. Une propriété vive a sa syntaxe et sa sémantique appliquées par le serveur. Les propriétés vives incluent des cas où a) la valeur d'une propriété est protégée et conservée par le serveur, et b) la valeur de la propriété est conservée par le client, mais le serveur effectue une vérification de syntaxe sur les valeurs soumises. Toutes les instances d'une propriété vive donnée DOIVENT se conformer à la définition associée à ce nom de propriété. Une propriété morte a sa syntaxe et sa sémantique appliquées par le client; le serveur enregistre simplement la valeur de la propriété telle qu'elle est.

4.2 Propriétés et en-têtes HTTP

Les propriétés existent déjà, dans un sens limité, dans les en-têtes de message HTTP. Cependant, dans les environnements d'auteurs répartis, un nombre relativement grand de propriétés est nécessaire pour décrire l'état d'une ressource, et les régler/retourner toutes par des en-têtes HTTP est inefficace. Donc, un mécanisme est nécessaire pour permettre à un principal d'identifier un ensemble de propriétés auxquelles le principal est intéressé et d'établir ou restituer juste ces propriétés.

4.3 Valeurs de propriété

La valeur d'une propriété est toujours un fragment XML (bien formé).

XML a été choisi parce que c'est un format de données flexible, auto descriptif, et structuré qui prend en charge des définitions de schéma riches, et parce que il accepte plusieurs jeux de caractères. La nature auto descriptive de XML permet que toute valeur de propriété soit étendue en ajoutant des éléments. Les clients ne vont pas avoir de défaillance quand ils rencontrent des extensions parce que ils vont quand même avoir les données spécifiées dans le schéma d'origine et DOIVENT ignorer les éléments qu'ils ne comprennent pas.

La prise en charge par XML de plusieurs jeux de caractères permet que toute propriété lisible par l'homme soit codée et lue dans un jeu de caractères familier à l'utilisateur. La prise en charge par XML de plusieurs langages humains, en utilisant l'attribut "xml: lang", traite des cas où le même jeu de caractères est employé par plusieurs langages humains. Noter que la portée de xml:lang est récurrente, de sorte qu'un attribut xml:lang sur tout élément contenant un nom de propriété s'applique à la valeur de la propriété sauf si elle a été écrasée par un attribut de portée plus locale. Noter qu'une propriété a seulement une valeur, dans un langage (ou le langage PEUT rester indéfini) ; une propriété n'a pas plusieurs valeurs dans des langages différents ou une seule valeur dans plusieurs langages.

Une propriété est toujours représentée avec un élément XML consistant en le nom de la propriété, appelée "élément nom de propriété". Le plus simple exemple est une propriété vide, qui est différente d'une propriété qui n'existe pas :

```
<R:title xmlns:R="http://www.exemple.com/ns/"></R:title>
```

La valeur de la propriété apparaît à l'intérieur de l'élément Nom de propriété. La valeur peut être toute sorte de contenu XML bien formé, incluant un contenu de texte seulement et mixte. Les serveurs DOIVENT préserver les éléments d'information XML suivants (en utilisant la terminologie de [XML-INFOSET]) dans la mémorisation et la transmission des propriétés mortes :

Pour l'élément d'information d'élément Nom de propriété lui-même :
[nom d'espace de noms]
[nom local]
[attributs] "xml:lang" désigné ou tout attribut de cette sorte dans la portée
[fils] d'élément ou caractère de type

Sur tout éléments Élément d'information dans la valeur de propriété :
[nom d'espace de noms]
[nom local]
[attributs]
[fils] d'élément ou caractère de type

Sur les éléments Informations d'attribut dans la valeur de propriété :
[nom d'espace de noms]
[nom local]
[valeur normalisée]

Sur les éléments Informations de caractère dans la valeur de propriété :
[code de caractère]

Comme les préfixes sont utilisés dans certains vocabulaires XML (par exemple, les schémas XPath et XML) les serveurs DEVRAIENT les préserver pour tout élément d'information dans la valeur [prefix].

Les attributs XML Infoset non mentionnés ci-dessus PEUVENT être préservés par le serveur, mais les clients NE DOIVENT PAS compter sur leur préservation. Les règles ci-dessus vont aussi s'appliquer par défaut aux propriétés vives, sauf mention contraire.

Les serveurs DOIVENT ignorer l'attribut XML xml:space si il est présent et ne jamais l'utiliser pour changer le traitement des espaces. Les espaces dans les valeurs de propriété sont significatives.

4.3.1 Exemple - Propriété avec contenu mixte

Considérons une propriété morte "auteur" créée par le client comme suit :

```
<D:prop xml:lang="fr" xmlns:D="DAV:">
  <x:auteur xmlns:x='http://exemple.com/ns'>
   <x:nom>Jane Doe</x:nom>
   <!-- Informations de contact de Jane -->
   <x:uri type='email'
        added='2005-11-26'>mailto:jane.doe@exemple.com</x:uri>
   <x:uri type='web'
        added='2005-11-27'>http://www.exemple.com</x:uri>
   <x:notes xmlns:h='http://www.w3.org/1999/xhtml'>
    Jane a travaillé <h:em>trop</h:em> longtemps sur la révision attendue de <![CDATA[<RFC2518>]]>.
   </x:notes>
  </x:auteur>
</D:prop>
```

Quand cette propriété est demandée, un serveur pourrait retourner :

```
<D:prop xmlns:D='DAV:'><auteur
      xml:lang='fr'
      xmlns:x='http://exemple.com/ns'
      xmlns='http://exemple.com/ns'
      xmlns:h='http://www.w3.org/1999/xhtml'>
   <x:name>Jane Doe</x:name>
   <x:uri   added="2005-11-26" type="email"
    >mailto:jane.doe@exemple.com</x:uri>
   <x:uri   added="2005-11-27" type="web"
    >http://www.exemple.com</x:uri>
   <x:notes>
    Jane a travaillé <h:em>trop</h:em> longtemps sur la révision attendue de &lt;RFC2518&gt;.
   </x:notes>
  </auteur>
</D:prop>
```

Noter que dans cet exemple :

- le [préfixe] pour le nom de propriété n'a pas été préservé, étant non significatif, tandis que toutes les autres valeurs de [préfixe] ont été préservées ;
- les valeurs d'attribut ont été réécrites avec des guillemets doubles à la place de simples (le style de guillemets n'est pas significatif) et l'ordre des attributs n'a pas été préservé ;
- l'attribut xml:lang a été retourné sur l'élément Nom de propriété lui-même (il était dans la portée quand la propriété a été établie, mais la position exacte dans la réponse n'est pas considérée comme significative tant qu'elle est dans la portée) ;
- l'espace entre les étiquettes a été préservée partout (mais pas les espaces entre attributs) ;
- l'encapsulation CDATA a été remplacée par un échappement de caractère (l'inverse serait aussi légal) ;
- l'élément de commentaire a été supprimé (comme l'aurait été un élément d'instruction de traitement).

Note de mise en œuvre : il y a des cas comme les scénarios d'édition où les clients peuvent exiger que le contenu XML soit préservé caractère par caractère (comme un ordre des attributs ou un style de guillemets). Dans ce cas, les clients

devraient envisager d'utiliser une valeur de propriété texte-seulement en échappant tous les caractères qui ont une signification spéciale dans l'analyse XML.

4.4 Noms de propriété

Un nom de propriété est un identifiant universellement unique, c'est-à-dire associé à un schéma qui fournit des informations sur la syntaxe et la sémantique de la propriété.

Parce que un nom de propriété est unique au monde, les clients peuvent compter sur un comportement cohérent pour une propriété particulière sur plusieurs ressources, sur le même et à travers des serveurs différents, tant que cette propriété est "active" sur les ressources en question, et la mise en œuvre de la propriété active est fidèle à sa définition.

Le mécanisme d'espace de noms XML, qui se fonde sur des URI [RFC3986], est utilisé pour nommer les propriétés parce que il prévient les collisions d'espaces de nom et fournit des degrés variés de contrôle administratif.

L'espace de noms de propriété est plat; c'est-à-dire, aucune hiérarchie des propriétés n'est reconnue explicitement. Donc, si une propriété A et une propriété A/B existent sur une ressource, il n'y a pas de reconnaissance de relation entre les deux propriétés. On suppose qu'une spécification distincte sera éventuellement produite pour traiter les problèmes relatifs aux propriétés hiérarchiques.

Finalement, il n'est pas possible de définir deux fois la même propriété sur une seule ressource, car cela causerait une collision dans l'espace de noms de propriété de la ressource.

4.5 Ressources de source et ressources de résultat

Certaines ressources sont générées dynamiquement par le serveur. Pour ces ressources, il existe probablement quelque part un code de source qui contrôle comment cette ressource est générée. La relation des fichiers de source aux ressources HTTP résultantes peut être biunivoque, d'une à plusieurs, de plusieurs à une, ou de plusieurs à plusieurs. Il n'y a pas de mécanisme dans HTTP pour déterminer si une ressource est dynamique, autonome, si ses fichiers sources existent, ou comment leur attribuer un auteur. Bien qu'il serait utile de résoudre ce problème, les mises en œuvre interopérables de WebDAV ont été largement déployées sans réellement le résoudre, en traitant seulement des ressources statiques. Donc, le problème de source contre résultat n'est pas résolu dans la présente spécification et a été renvoyé à un document distinct.

5. Collections de ressources de la Toile

Cette Section donne une description d'un type de ressource de la Toile, la collection, et discute de ses interactions avec l'espace de noms d'URL HTTP et avec les méthodes HTTP. L'objet d'une ressource de collection est de modéliser des objets de style collection (par exemple, des répertoires de systèmes de fichiers) au sein de l'espace de noms d'un serveur.

Toutes les ressources conformes à DAV DOIVENT prendre en charge le modèle d'espace de noms d'URI HTTP spécifié ici.

5.1 Modèle d’espace de noms d’URL HTTP

L'espace de noms d'URL HTTP est hiérarchique et la hiérarchie est délimitée avec le caractère "/".

Un espace de noms d'URL HTTP est dit être cohérent si il satisfait les conditions suivantes : pour chaque URL dans la hiérarchie HTTP il existe une collection qui contient cet URL comme membre interne de l'URL. La racine, ou collection de niveau supérieur de l'espace de noms considéré, est exempte de cette règle. La collection de l'espace de noms considéré n'est pas nécessairement la collection identifiée par le chemin absolu '/' -- elle peut être identifiée par un ou plusieurs segments de chemin (par exemple, /servlets/webdav/...)

Ni HTTP/1.1 ni WebDAV n'exigent que l'espace de noms d'URL HTTP entier soit cohérent -- une ressource compatible WebDAV peut ne pas avoir de collection parente. Cependant, il est interdit à certaines méthodes WebDAV de produire des résultats qui causent des incohérences d'espace de noms.

Comme il est dit implicitement dans les [RFC2616] et [RFC3986], toute ressource, incluant les ressources de collection de collection, PEUT être identifiée par plus d'un URI. Par exemple, une ressource pourrait être identifiée par plusieurs URL HTTP.

5.2 Ressources de collection

Les ressources de collection diffèrent des autres ressources en ce qu'elles agissent aussi comme des conteneurs. Certaines méthodes HTTP s'appliquent seulement à une collection, mais certaines s'appliquent à une partie ou à toutes les ressources à l'intérieur du conteneur défini par la collection. Quand la portée d'une méthode n'est pas claire, le client peut spécifier quelle profondeur appliquer. La profondeur peut être de niveau zéro (seulement la collection), de niveau un (la collection et les ressources directement contenues) ou de niveaux infinis (la collection et toutes les ressources contenues de façon récurrente).

L'état d'une collection consiste en au moins un ensemble de transpositions entre segments de chemin et ressources, et un ensemble de propriétés sur la collection elle-même. Dans le présent document, une ressource B sera dite être contenue dans la ressource de collection A si il y a une transposition de segment de chemin qui transpose en B et c'est-à-dire est contenue dans A. Une collection DOIT contenir au plus une transposition pour un segment de chemin donné, c'est-à-dire, il est illégal d'avoir le même segment de chemin transposé en plus d'une ressource.

Les propriétés définies sur les collections se comportent exactement comme le font les propriétés sur des ressources non de collection. Une collection PEUT avoir un état supplémentaire comme corps d'entité retourné par GET.

Pour toutes les ressources A et B conformes à WebDAV, identifiées par les URL "U" et "V", respectivement, tels que "V" soit égal à "U/SEGMENT", A DOIT être une collection qui contient une transposition de "SEGMENT" à B. Donc, si la ressource B avec l'URL "http://exemple.com/bar/blah" est conforme à WebDAV et si la ressource A avec l'URL "http://exemple.com/bar/" est conforme à WebDAV, alors la ressource A doit être une collection et doit contenir exactement une transposition de "blah" en B.

Bien que généralement une transposition consiste en un seul segment et une ressource, en général, une transposition consiste en un ensemble de segments et une ressource. Cela permet à un serveur de traiter un ensemble de segments comme équivalents (c'est-à-dire, soit tous les segments sont transposés en la même ressource, soit aucun des segments n'est transposé en une ressource). Par exemple, un serveur qui effectue le repliement de casse sur les segments va traiter les segments "ab", "Ab", "aB", et "AB" comme équivalents. Un client peut alors utiliser un de ces segments pour identifier la ressource. Noter qu'un résultat PROPFIND va choisir un de ces segments équivalents pour identifier la transposition, de sorte qu'il va y avoir un élément de réponse PROPFIND par transposition, et non un par segment dans la transposition.

Les ressources de collection PEUVENT avoir des transpositions à des ressources non conformes à WebDAV dans la hiérachie d'espace de noms d'URL HTTP mais elles ne sont pas obligées de faire ainsi. Par exemple, si la ressource X avec l'URL "http://exemple.com/bar/blah" n'est pas conforme à WebDAV et si la ressource A avec l'URL "http://exemple.com/bar/" identifie une collection WebDAV, alors A peut ou non avoir une transposition de "blah" en X.

Si une ressource conforme à WebDAV n'a pas de membre interne conforme à WebDAV dans la hiérarchie d'espace de noms d'URL HTTP, alors il n'est pas obligé que la ressource conforme à WebDAV soit une collection.

Il existe la convention que quand il est fait référence à une collection par son nom sans barre oblique en queue, le serveur PEUT traiter la demande comme si la barre oblique en queue était présente. Dans ce cas, il DEVRAIT retourner un en-tête Content-Location dans la réponse, pointant sur l'URL en finissant par le "/". Par exemple, si un client invoque une méthode sur http://exemple.com/blah (pas de barre oblique en queue) le serveur peut répondre comme si l'opération était invoquée sur http://exemple.com/blah/ (avec barre oblique en queue) et devrait retourner un en-tête Content-Location avec la valeur http://exemple.com/blah/. Chaque fois qu'un serveur produit un URL se référant à une collection, le serveur DEVRAIT inclure la barre oblique en queue. En général, les clients DEVRAIENT utiliser la forme avec barre oblique en queue des noms de collection. Si les clients n'utilisent pas la forme de barre oblique en queue, le client doit s'attendre à voir une réponse de redirection. Les clients vont trouver la propriété DAV:resourcetype plus fiable que l'URL pour trouver si une ressource est une collection.

Les clients DOIVENT être capables de prendre en charge le cas où les ressources sont contenues dans des ressources non WebDAV. Par exemple, si une réponse OPTIONS provenant de "http://exemple.com/servlet/dav/collection" indique la prise en charge de WebDAV, le client ne peut pas supposer que "http://exemple.com/servlet/dav/" ou son parent sont nécessairement des collections WebDAV.

Un scénario normal dans lequel les URL transposés n'apparaissent pas comme membres de leur collection parente est le cas où un serveur permet des liaisons ou des redirections sur des ressources non WebDAV. Par exemple, "/col/link" pourrait ne pas apparaître comme un membre de "/col/", bien que le serveur répondrait par un état 302 à une demande GET à "/col/link" ; donc, l'URL "/col/link" va bien sûr être transposé. De même, une page générée dynamiquement pourrait avoir un URL transposant de "/col/index.html", donc cette ressource pourrait répondre par un 200 OK à une demande GET bien qu'elle n'apparaisse pas comme un membre de "/col/".

Certaines transpositions à des ressources même conformes à WebDAV pourraient ne pas apparaître dans la collection parente. Un exemple de ce cas est celui des serveurs qui acceptent plusieurs alias d'URL pour chaque ressource conforme à WebDAV. Un serveur peut mettre en œuvre des URL insensibles à la casse, où "/col/a" et "/col/A" identifient la même ressource, mais seulement "a" ou "A" est rapporté dans la liste des membres de "/col". Dans les cas où un serveur traite un ensemble de segments comme équivalents, le serveur DOIT exposer seulement un segment préféré par transposition, choisi de façon cohérente, dans les réponses PROPFIND.

6. Verrouillage

La capacité de verrouiller une ressource fournit un mécanisme pour mettre en série l'accès à cette ressource. En utilisant un verrou, un client d'auteur peut fournir une garantie raisonnable qu'un autre principal ne va pas modifier une ressource pendant qu'elle est en cours d'édition. De cette façon, un client peut empêcher le problème de la "mise à jour perdue".

La présente spécification permet que les verrous varient entre deux paramètres spécifiés par le client, selon le nombre de principaux impliqués (exclusif ou partagé) et le type d'accès à accorder. Le présent document définit le verrouillage pour un seul type d'accès, écriture. Cependant, la syntaxe est extensible, et permet la spécification éventuelle de verrouillage pour d'autres types d'accès.

6.1 Modèle de verrou

Ce paragraphe fournit un modèle concis du comportement de verrouillage. Les paragraphes suivants donnent plus de détails sur certains des concepts et renvoient à ces déclarations de modèle. Les déclarations normatives relatives au traitement des méthodes LOCK et UNLOCK se trouvent dans les paragraphes qui décrivent ces méthodes, tandis que les déclarations normatives qui couvrent toute méthode sont rassemblées ici.

1. Un verrou verrouille une ressource directement ou indirectement.

2. Une ressource devient directement verrouillée quand une demande LOCK à un URL de cette ressource crée un nouveau verrou. Le "lock-root" *(verrou racine)* du nouveau verrou est cet URL. Si au moment de la demande, l'URL n'est pas transposé en une ressource, une nouvelle ressource vide est créée et directement verrouillée.

3. Un verrou exclusif (paragraphe 6.2) entre en conflit avec toute autre sorte de verrou sur la même ressource, que l'un ou l'autre verrou soit direct ou indirect. Un serveur NE DOIT PAS créer de verrous en conflit sur une ressource.

4. Pour une collection c'est-à-dire verrouillée avec un verrou L de profondeur infinie, toutes les ressources membres sont indirectement verrouillées. Les changements des membres d'une telle collection affectent l'ensemble des ressources verrouillées indirectement :
   * Si une ressource membre est ajoutée à la collection, la nouvelle ressource membre NE DOIT PAS avoir déjà un verrou en conflit, parce que la nouvelle ressource DOIT devenir indirectement verrouillée par L.
   * Si une ressource membre cesse d'être membre de la collection, la ressource DOIT alors ne plus être indirectement verrouillée par L.

5. Chaque verrou est identifié par un seul jeton de verrou unique au monde (paragraphe 6.5).

6. Une demande UNLOCK supprime le verrou avec le jeton de verrou spécifié. Après la suppression d'un verrou, aucune ressource n'est verrouillée par ce verrou.

7. Un jeton de verrou est "soumis" dans une demande quand il apparaît dans un en-tête "If" (la Section 7, "Verrou en écriture", discute de quand la soumission de jeton est exigée pour les verrous en écriture).

8. Si une demande est cause que le verrou racine de tout verrou devient un URL non transposé, le verrou DOIT alors aussi être supprimé par cette demande.

6.2 Verrou exclusif ou partagé

La forme la plus basique de verrou est un verrou exclusif. Les verrous exclusifs évitent d'avoir à traiter des conflits de changement de contenu, sans exiger de coordination autre que les méthodes décrites dans la présente spécification.

Cependant, lorsque le but d'un verrou n'est pas d'exclure d'autres personnes d'exercer un droit d'accès mais plutôt de fournir un mécanisme pour que les principaux indiquent qu'ils ont l'intention d'exercer leurs droits d'accès. Les verrous partagés

sont fournis pour ce cas. Un verrou partagé permet à plusieurs principaux de recevoir un verrou. Donc, tout principal qui a à la fois les privilèges d'accès et un verrou valide peut utiliser la ressource verrouillée.

Avec les verrous partagés, il y a deux ensembles de confiance qui affectent une ressource. Le premier ensemble de confiance est créé par des permissions d'accès. Les principaux qui sont de confiance, par exemple, peuvent avoir la permission d'écrire sur la ressource. Parmi ceux qui ont la permission d'accès en écriture sur la ressource, l'ensemble des principaux qui ont sorti un verrou partagé doivent aussi se faire mutuellement confiance, en créant un (normalement) plus petit ensemble de confiance au sein de l'ensemble de ceux qui ont la permission d'accès en écriture.

En commençant avec tous les principaux possibles dans l'Internet, dans la plupart des situations, la vaste majorité de ces principaux n'auront pas d'accès en écriture sur une ressource donnée. Dans le petit nombre de ceux qui ont l'accès en écriture, certains principaux peuvent décider de garantir que leurs éditions sont libres de conflit d'écrasement en utilisant des verrous en écriture exclusifs. D'autres peuvent décider qu'ils font confiance à leurs collaborateurs pour ne pas écraser leur travail (l'ensemble potentiel de collaborateurs étant l'ensemble des principaux qui ont la permission d'écriture) et utiliser un verrou partagé, qui informe les collaborateurs qu'un principal peut être en train de travailler sur la ressource.

Les extensions WebDAV à HTTP n'ont pas besoin de fournir tous les chemins de communications nécessaires pour que les principaux coordonnent leurs activités. Quand ils utilisent des verrous partagés, les principaux peuvent utiliser tout canal de communication hors bande pour coordonner leur travail (par exemple, des interactions en face à face, des notes écrites, des notes sur l'écran, une conversation téléphonique, un message électronique , etc.) L'intention d'un verrou partagé est de faire connaître aux collaborateurs qui d'autre peut être en train de travailler sur une ressource.

Les verrous partagés sont inclus parce que l'expérience des systèmes d'auteurs distribués sur la Toile a indiqué que les verrous exclusifs sont souvent trop rigides. Un verrou exclusif est utilisé pour appliquer un processus d'édition particulier : sortir un verrou exclusif, lire la ressource, effectuer l'édition, écrire la ressource, libérer le verrou. Ce processus d'édition pose le problème que les verrous ne sont pas toujours correctement libérés, par exemple, quand un programme est défaillant ou quand un créateur de verrou laisse une ressource sans la déverrouiller. Bien que des temporisations (paragraphe 6.6) et des actions administratives puissent être utilisées pour retirer un verrou inapproprié, aucun de ces mécanismes peut n'être disponible quand on en a besoin ; la temporisation peut être longue ou l'administrateur peut n'être pas disponible.

La réussite d'une demande de nouveau verrou partagé DOIT résulter en la génération d'un verrou unique associé au principal demandeur. Donc, si cinq principaux ont sorti des verrous partagés en écriture sur la même ressource, il va y avoir cinq verrous et cinq jetons de verrou, un pour chaque principal.

6.3 Prise en charge exigée

Une ressource conforme à WebDAV n'est pas obligée de prendre en charge une forme quelconque de verrouillage. Si la ressource prend en charge le verrouillage, elle peut choisir de prendre en charge toute combinaison de verrous exclusifs et partagés pour tous types d'accès.

La raison de cette souplesse est que la politique de verrouillage touche au cœur des systèmes de gestion et de versions de ressources employés par les divers répertoires de mémorisation. Ces répertoires exigent le contrôle de la sorte de verrouillage qui va être rendu disponible. Par exemple, certains répertoires prennent seulement en charge les verrous partagés en écriture, tandis que d'autres fournissent seulement la prise en charge de verrous en écriture exclusifs, et que d'autres n'utilisent pas de verrouillage du tout. Comme chaque système est suffisamment différent pour mériter l'exclusion de certaines caractéristiques de verrouillage, la présente spécification garde le verrouillage comme seul axe de négociation au sein de WebDAV.

6.4 Créateur de verrou et privilèges

Le créateur d'un verrou a des privilèges particuliers pour utiliser le verrou et modifier la ressource. Quand une ressource verrouillée est modifiée, un serveur DOIT vérifier que le principal authentifié correspond au créateur du verrou (en plus de la vérification d'une soumission valide de jeton de verrou).

Le serveur PEUT permettre aux utilisateurs privilégiés autres que le créateur du verrou de détruire un verrou (par exemple, le propriétaire de la ressource ou un administrateur). Le privilège 'unlock' dans la [RFC3744] a été défini pour fournir cette permission.

Il n'est pas exigé que les serveurs acceptent les demandes LOCK provenant de tous les utilisateurs ou d'utilisateurs anonymes.

Noter qu'avoir un verrou ne confère pas le privilège de modifier la ressource verrouillée. L'accès en écriture et les autres privilèges DOIVENT être appliqués par les mécanismes normaux de privilège ou d'authentification, et non sur la base de la possible obscurité des valeurs de jeton de verrou.

6.5 Jetons de verrou

Un jeton de verrou est un type de jeton d'état qui identifie un verrou particulier. Chaque verrou a exactement un unique jeton de verrou généré par le serveur. Les clients NE DOIVENT PAS tenter d'interpréter de quelque façon que ce soit les jetons de verrou.

Les URI de jeton de verrou DOIVENT être uniques sur toutes les ressources tout le temps. Cette contrainte d'unicité permet aux jetons de verrou d'être soumis à travers les ressources et serveurs sans crainte de confusion. Comme les jetons de verrou sont uniques, un client PEUT soumettre un jeton de verrou dans un en-tête If sur une ressource autre que celle qui l'a retourné.

Quand une opération LOCK crée un nouveau verrou, le nouveau jeton de verrou est retourné dans l'en-tête de réponse Lock-Token défini au paragraphe 10.5, et aussi dans le corps de la réponse.

Les serveurs PEUVENT rendre les jetons de verrou lisibles publiquement (par exemple, dans la propriété DAV: lockdiscovery). Un cas d'utilité de la lisibilité du jeton de verrou est qu'un verrou de longue durée de vie puisse être retiré par le propriétaire de la ressource (le client qui a obtenu le verrou pourrait être défaillant ou déconnecté avant d'avoir pu nettoyer le verrou). Sauf pour le cas de l'utilisation de UNLOCK sous le contrôle de l'utilisateur, un client NE DEVRAIT PAS utiliser un jeton de verrou créé par une autre instance de client.

La présente spécification encourage les serveurs à créer des identifiants univoques universels (UUID, *Universally Unique Identifier*) pour les jetons de verrou, et d'utiliser la forme d'URI définie par l'espace de noms d'URN d'identifiant univoque universel ([RFC4122]). Cependant, les serveurs sont libres d'utiliser tout URI (par exemple, provenant d'un autre schéma) pour autant qu'il satifasse à l'exigence d'unicité. Par exemple, un jeton de verrou valide pourrait être construit en utilisant le schéma "opaquelocktoken" défini à l'Appendice C.

Exemple : "urn:uuid:f81d4fae-7dec-11d0-a765-00a0c91e6bf6"

6.6 Temporisation de verrou

Un verrou PEUT avoir une durée de vie limitée. La durée de vie est suggérée par le client quand il crée ou rafraîchit le verrou, mais le serveur choisit en fin de compte la valeur de la temporisation. La temporisation est mesurée en secondes restantes jusqu'à l'expiration du verrou.

Le compteur de temporisation DOIT être redémarré si une demande de rafraîchissement de verrou est réussie (voir au paragraphe 9.10.2). Le compteur de temporisation NE DEVRAIT PAS être redémarré à un autre moment.

Si le temporisateur expire, le verrou DEVRAIT alors être supprimé. Dans ce cas, le serveur DEVRAIT agir comme si une méthode UNLOCK était exécutée par le serveur sur la ressource en utilisant le jeton de verrou du verrou expiré, effectuée avec son autorité d'outrepassement.

Il est conseillé aux serveurs de faire très attention aux valeurs soumises par les clients, car elles vont être des indications du type d'activité que le client entend effectuer. Par exemple, un sous programme fonctionnant sur un navigateur peut avoir besoin de verrouiller une ressource, mais à cause de l'instabilité de l'environnement du fonctionnement du sous programme, celui-ci peut être désactivé sans avertissement. Par suite, le sous programme va probablement demander une valeur de temporisation relativement courte afin que si le sous programme s'arrête, le verrou puisse être rapidement retiré. Cependant, un système de gestion de document va probablement demander une temporisation extrêmement longue parce que un utilisateur peut prévoir de passer hors ligne.

Un client NE DOIT PAS supposer que juste parce que le temporisateur a expiré, le verrou va être immédiatement retiré.

De même, un client NE DOIT PAS supposer que juste parce que le temporisateur n'a pas expiré, le verrou existe encore. Les clients DOIVENT supposer que les verrous peuvent disparaître arbitrairement à tout moment, sans considération de la valeur donnée dans l'en-tête Timeout. L'en-tête Timeout indique seulement le comportement du serveur si des circonstances extraordinaires ne se produisent pas. Par exemple, un utilisateur suffisamment privilégié peut supprimer un verrou à tout moment, ou le système peut avoir une défaillance telle qu'il perde l'enregistrement de l'existence du verrou.

6.7 Découverte de capacité de verrou

Comme la prise en charge des verrous par le serveur est facultative, un client qui essaye de verrouiller une ressource sur un serveur peut soit essayer le verrou et espérer que cela fonctionne, soit effectuer une forme de découverte pour déterminer quelles capacités de verrouillage le serveur prend en charge. C'est ce qu'on appelle la découverte de capacité de verrouillage. Un client peut déterminer quels types de verrouillage le serveur prend en charge en restituant la propriété DAV:supportedlock.

Toute ressource conforme à DAV qui prend en charge la méthode LOCK DOIT prendre en charge la propriété DAV:supportedlock *(verrou pris en charge par DAV)*.

6.8 Découverte active de verrou

Si un autre principal verrouille une ressource à laquelle un principal souhaite accéder, il est utile au second principal d'être capable de trouver qui est le premier principal. À cette fin, la propriété DAV:lockdiscovery *(découverte de verrou DAV)* est fournie. Cette propriété fait la liste de tous les verrous en cours, décrit leur type, et PEUT même fournir les jetons de verrous.

Toute ressource conforme à DAV qui prend en charge la méthode LOCK DOIT prendre en charge la propriété DAV:lockdiscovery.

7. Verrou en écriture

Cette Section décrit la sémantique spécifique du type de verrou en écriture. Le verrou en écriture est une instance spécifique d'un type de verrou, et est le seul type de verrou décrit dans la présente spécification.

Un verrou en écriture exclusif protège une ressource : il empêche les changements par tout principal autre que le créateur du verrou et dans tous les cas où le jeton de verrou n'est pas soumis (par exemple, par un processus de client autre que de celui qui détient le verrou).

Les clients DOIVENT soumettre un jeton de verrou qu'ils sont autorisés à utiliser dans toute demande qui modifie une ressource verrouillée en écriture. La liste des modifications couvertes par un verrou en écriture inclut :

1. Un changement d'un des aspects suivants de toute ressource verrouillée en écriture :
   * toute variante,
   * toute propriété morte,
   * toute propriété vive c'est-à-dire verrouillable (une propriété vive est verrouillable sauf définie autrement.)

2. Pour les collections, toute modification d'un URI de membre interne. Un URI de membre interne d'une collection est considéré être modifié si il est ajouté, supprimé, ou identifie une ressource différente. On trouvera une discussion plus détaillée sur les verrous en écriture et les collections au paragraphe 7.4.

3. Une modification de la transposition de la racine du verrou en écriture, soit sur une autre ressource, soit pas de ressource (par exemple, DELETE).

Parmi les méthodes définies dans HTTP et WebDAV, PUT, POST, PROPPATCH, LOCK, UNLOCK, MOVE, COPY (pour la ressource de destination) DELETE, et MKCOL sont affectées par les verrous en écriture. Toutes les autres méthodes HTTP/WebDAV définies jusqu'ici -- GET en particulier -- fonctionnent indépendamment d'un verrou en écriture.

Les paragraphes qui suivent décrivent en termes plus spécifiques comment les verrous en écriture interagissent avec les diverses opérations.

7.1 Verrous en écriture et propriétés

Bien que celles sans verrou en écriture ne puissent pas altérer une propriété sur une ressource, il est quand même possible que les valeurs de propriétés vives changent, même quand elles sont verrouillées, du fait des exigences de leurs schémas. Seules les propriétés mortes et les propriétés vives définies comme verrouillables ont la garantie de ne pas changer quand elles sont verrouillées en écriture.

7.2 Éviter les pertes de mise à jour

Bien que les verrous en écriture apportent une certaine aide pour empêcher les pertes de mises à jour, ils ne peuvent pas garantir que des mises à jour ne seront jamais perdues. On examine le scénario suivant :
Deux clients A et B sont intéressés à l'édition de la ressource 'index.html'. Le client A est un client HTTP plutôt que WebDAV, et donc ne sait pas comment effectuer le verrouillage. Le client A ne verrouille pas le document, mais fait un GET, et commence l'édition. Le client B fait un LOCK, effectue un GET et commence l'édition. Le client B termine son édition, effectue un PUT, puis un UNLOCK. Le client A effectue un PUT, écrasant et perdant tous les changements de B.

Il y a plusieurs raisons au fait que le protocole WebDAV lui-même ne peut pas empêcher cette situation. D'abord, il ne peut pas forcer tous les clients à utiliser le verrouillage parce que il doit être compatible avec les clients HTTP qui ne comprennent pas le verrouillage. Ensuite, il ne peut pas exiger des serveurs qu'ils prennent en charge le verrouillage à cause de la diversité des mises en œuvre de répertoires, dont certaines s'appuient sur des réservations et des fusions plutôt que sur le verrouillage. Finalement, étant sans état, il ne peut pas appliquer une séquence d'opérations comme LOCK/GET/PUT/UNLOCK.

Les serveurs WebDAV qui prennent en charge le verrouillage peuvent réduire la probabilité que les clients écrasent accidentellement les changements des autres en exigeant que les clients verrouillent les ressources avant de les modifier. Ces serveurs vont effectivement empêcher les clients HTTP 1.0 et HTTP 1.1 de modifier les ressources.

Les clients WebDAV peuvent être de bons citoyens en utilisant une séquence d'opérations verrouillage / restitution / écriture / déverrouillage (au moins par défaut) chaque fois qu'ils interagissent avec un serveur WebDAV qui prend en charge le verrouillage.

Les clients HTTP 1.1 peuvent être de bons citoyens, évitant d'écraser les changements des autres clients, en utilisant des étiquettes d'entité dans les en-têtes If-Match avec toutes les demandes qui vont modifier des ressources.

Les gestionnaires d'informations peuvent tenter d'empêcher les écrasements en mettant en œuvre des procédures côté client qui exigent le verrouillage avant la modification de ressources WebDAV.

7.3 Verrous en écriture et URL non transposés

WebDAV fournit la capacité d'envoyer une demande LOCK à un URL non transposé afin de réserver le nom à utiliser. C'est une façon simple d'éviter le problème de la mise à jour perdue sur la création d'une nouvelle ressource (une autre façon est d'utiliser l'en-tête If-None-Match spécifié au paragraphe 14.26 de la [RFC2616]). Cela a l'avantage collatéral de verrouiller la nouvelle ressource immédiatement pour qu'elle soit utilisée par le créateur.

Noter que le problème de la mise à jour perdue ne se pose pas pour les collections parce que MKCOL peut seulement être utilisé pour créer une collection, non pour écraser une collection existante. Quand ils essayent de verrouiller une collection à la création, les clients peuvent tenter d'augmenter la probabilité d'obtenir le verrouillage en traitant en parallèle des demandes MKCOL et LOCK (mais parce que cela ne convertit pas deux opérations séparés en une opération atomique, il n'est pas garanti que cela fonctionne).

Une demande de verrouillage réussie à un URL non transposé DOIT résulter en la création d'une ressource verrouillée (non de collection) avec un contenu vide. Ensuite, une demande PUT réussie (avec le jeton de verrou correct) fournit le contenu de la ressource. Noter que la demande LOCK n'a pas de mécanisme pour que le client fournisse un Content-Type ou Content-Language, donc le serveur va utiliser des valeurs par défaut ou vides et s'appuyer sur la demande PUT suivante pour les valeurs correctes.

Une ressource créée avec un LOCK est vide mais se comporte par ailleurs de toutes façons comme une ressource normale. Elle se comporte de la même façon qu'une ressource créée par une demande PUT avec un corps vide (et où un Content-Type et Content-Language n'ont pas été spécifiés) suivie par une demande LOCK à la même ressource. Suivant ce modèle, une ressource verrouillée vide :

- Peut être lue, supprimée, déplacée, et copiée, et se comporte de toutes les façons comme une ressource régulière non de collection.
- Apparaît comme un membre de sa collection parente.
- NE DEVRAIT PAS disparaître quand son verrou s'en va (les clients doivent donc être responsables du nettoyage de leurs propres affaires, comme avec toute autre opération ou toute ressource non vide).
- NE PEUT PAS avoir des valeurs pour des propriétés comme DAV:getcontentlanguage qui n'ont pas encore été spécifiées par le client.
- PEUT être mise à jour (avoir un contenu ajouté) avec une demande PUT.
- NE DOIT PAS être convertie en une collection. Le serveur DOIT faire échouer une demande MKCOL (car ce serait une

demande MKCOL pour toute ressource existante non de collection).
- DOIT avoir des valeurs définies pour les propriétés DAV:lockdiscovery et DAV: supportedlock.
- La réponse DOIT indiquer qu'une ressource a été créée, en utilisant le code de réponse "201 Créée" (une demande LOCK à une ressource existante résulterait en un 200 OK). Le corps doit quand même inclure la propriété DAV:lockdiscovery, comme avec une demande LOCK à une ressource existante.

Le client est supposé mettre à jour la ressource verrouillée vide peu après l'avoir verrouillée, en utilisant PUT et éventuellement PROPPATCH.

Autrement et pour la rétro compatibilité avec la [RFC2518], les serveurs PEUVENT mettre en œuvre à la place des ressources verrouillées vides (LNR, *Lock-Null Resource*) (voir la définition à l'Appendice D). Les clients peuvent facilement interopérer avec les serveurs qui prennent en charge les LNR d'ancien modèle et le modèle recommandé de "ressource verrouillée vide" en essayant seulement PUT après un LOCK sur un URL non transposé, pas MKCOL ou GET, et en ne comptant pas sur des propriétés spécifiques des LNR.

7.4 Verrous en écriture et collections

Il y a deux sortes de collections de verrous en écriture. Un verrou en écriture de profondeur 0 sur une collection protège les propriétés de collection plus les URL de membres internes de cette collection, tout en ne protégeant pas le contenu ou les propriétés des ressources membres (si la collection elle-même n'a pas de corps d'entité, elles sont aussi protégées). Un verrou en écriture de profondeur infinie sur une collection fournit la même protection sur cette collection et fournit aussi la protection de verrou en écriture sur chaque ressource membre.

Exprimé autrement, un verrou en écriture de l'une ou l'autre sorte protège toute demande qui créerait une nouvelle ressource dans une collection verrouillée en écriture, toute demande qui supprimerait un URL de membre interne d'une collection verrouillée en écriture, et toute demande qui changerait le nom de segment de tout membre interne.

Donc, un verrou en écriture de collection protège toutes les actions suivantes :
- DELETE *(supprime)* un membre interne direct d'une collection,
- MOVE *(déplace)* un membre interne hors de la collection,
- MOVE insérant un nouveau membre interne dans la collection,
- MOVE pour changer le nom d'un membre interne dans une collection,
- COPY *(copie)* un membre interne dans une collection, et
- une demande PUT ou MKCOL qui créerait un nouveau membre interne.

Le jeton de verrou de la collection est exigé en plus du jeton de verrou sur le membre interne lui-même, si il est verrouillé séparément.

De plus, un verrou de profondeur infinie affecte toutes les opérations d'écriture de tous les membres de la collection verrouillée. Avec un verrou de profondeur infinie, la ressource identifiée par la racine du verrou est directement verrouillée, et tous ses membres sont indirectement verrouillés.

- Toute nouvelle ressource ajoutée comme descendant d'une collection verrouillée de profondeur infinie devient indirectement verrouillée.
- Toute ressource indirectement verrouillée déplacée hors de la collection verrouillée dans une collection non verrouillée est à partir de là non verrouillée.
- Toute ressource indirectement verrouillée déplacée hors d'une collection source verrouillée dans une collection cible à verouillage de profondeur infinie reste indirectement verrouillée mais est maintenant protégée par le verrou sur la collection cible (le jeton de verrou de la collection cible va à partir de là être obligé de faire d'autres changements).

Si une demande LOCK en écriture de profondeur infinie est produite sur une collection contenant des URL membres identifiant des ressources qui sont actuellement verrouillées d'une manière qui entre en conflit avec le nouveau verrou (voir au paragraphe 6.1, point 3) la demande DOIT échouer avec un code d'état 423 (Verrouillé) et la réponse DEVRAIT contenir la précondition 'no-conflicting-lock' *(pas de verrou en conflit)*.

Si une demande de verrou cause l'ajout de l'URL d'une ressource à un URL de membre interne d'une collection verrouillée à la profondeur infinie, alors la nouvelle ressource DOIT être automatiquement protégée par le verrou. Par exemple, si la collection /a/b/ est verrouillée en écriture et si la ressource /c est déplacée à /a/b/c, alors la ressource /a/b/c va être ajoutée au verrou en écriture.

7.5 Verrous en écriture et en-tête de demande If

Un agent d'utilisateur doit démontrer sa connaissance d'un verrou quand il demande une opération sur une ressource verrouillée. Autrement, le scénario suivant pourrait se produire. Dans ce scénario, le programme A, de l'usager A, ôte un verrou en écriture sur une ressource. Le programme B, lui aussi de l'usager A, n'a pas connaissance du verrou oté par le programme A, et effectue un PUT sur la ressource verrouillée. Dans ce scénario, le PUT réussit parce que les verrous sont associés à un principal, et non à un programme, et donc le programme B, parce que il agit avec l'accréditif du principal A, a la permission d'effectuer le PUT. Cependant, si le programme B avait connu l'existence du verrou, il n'aurait pas écrasé la ressource, préférant plutôt présenter une boîte de dialogue décrivant le conflit à l'usager. Du fait de ce scénario, un mécanisme est nécessaire pour empêcher différents programmes d'ignorer accidentellement les verrous ôtés par d'autres programmes avec la même autorisation.

Afin d'empêhcer ces collisions, un jeton de verrou DOIT être soumis par un principal autorisé pour toutes les ressources verrouillées qu'une méthode peut changer, ou la méthode DOIT échouer. Un jeton de verrou est soumis quand il apparaît dans un en-tête If. Par exemple, si une ressource est à déplacer et si la source et la destination sont toutes deux verrouillées, alors deux jetons de verrou doivent être soumis dans l'en-tête If, un pour la source et l'autre pour la destination.

7.5.1 Exemple – Verrou en écriture et COPY

>>Demande

```
COPY /~fielding/index.html HTTP/1.1
Host: www.exemple.com
Destination: http://www.exemple.com/users/f/fielding/index.html
If: <http://www.exemple.com/users/f/fielding/index.html>
   (<urn:uuid:f81d4fae-7dec-11d0-a765-00a0c91e6bf6>)
```

>>Réponse

```
HTTP/1.1 204 Pas de contenu
```

Dans cet exemple, bien que la source et la destination soient toutes deux verrouillées, seulement un jeton de verrou doit être soumis (celui pour le verrou sur la destination). C'est parce que la ressource source n'est pas modifiée par un COPY, et donc non affectée par le verrou en écriture. Dans cet exemple, l'authentification de l'agent d'utilisateur est intervenue précédemment via un mécanisme qui sort du domaine d'application du protocole HTTP, dans la couche de transport sous-jacente.

7.5.2 Exemple – Suppression d'un membre d'une collection verrouillée

Considérons une collection "/verrouillé" avec un verrou en écriture exclusif, de profondeur infinie, et une tentative de suppression d'un membre interne "/verrouillé/membre" :

>>Demande

```
DELETE /verrouillé/membre HTTP/1.1
Host: exemple.com
```

>>Réponse

```
HTTP/1.1 423 Locked
Content-Type: application/xml; charset="utf-8"
Content-Length: xxxx

<?xml version="1.0" encoding="utf-8" ?>
<D:error xmlns:D="DAV:">
  <D:lock-token-submitted>
    <D:href>/verrouillé/</D:href>
  </D:lock-token-submitted>
</D:error>
```

Donc, le client va devoir soumettre le jeton de verrou avec la demande pour qu'elle réussisse. Pour cela, diverses formes de l'en-tête If (voir au paragraphe 10.4) pourraient être utilisées.

Format "No-Tag-List" :

If: (<urn:uuid:150852e2-3847-42d5-8cbe-0f4f296f26cf>)

Format "Tagged-List", pour "http://exemple.com/verrouillé/":

If: <http://exemple.com/verrouillé/>
(<urn:uuid:150852e2-3847-42d5-8cbe-0f4f296f26cf>)

Format "Tagged-List", pour "http://exemple.com/verrouillé/membre":

If: <http://exemple.com/verrouillé/membre>
(<urn:uuid:150852e2-3847-42d5-8cbe-0f4f296f26cf>)

Noter que, pour les besoins de la soumission du jeton de verrou, la forme réelle n'a pas d'importance; ce qui est pertinent est que le jeton de verrou apparaisse dans l'en-tête If, et que l'en-tête If lui-même s'évalue comme vrai.

7.6 Verrous en écriture et COPY/MOVE

Une invocation de méthode COPY NE DOIT PAS dupliquer de verrous en écriture actifs sur la source. Cependant, comme noté précédemment, si le COPY copie la ressource dans une collection, c'est-à-dire verrouillée avec un verrou de profondeur infinie, alors la ressource va être ajoutée au verrou.

Une demande MOVE réussie sur une ressource verrouillée en écriture NE DOIT PAS déplacer le verrou en écriture avec la ressource. Cependant, si il existe un verrou à la destination, le serveur DOIT ajouter la ressource déplacée à la portée du verrou de destination. Par exemple, si le MOVE fait de la ressource un enfant d'une collection qui a un verrou de profondeur infinie, alors la ressource va être ajoutée au verrou de cette collection. De plus, si une ressource avec un verrou de profondeur infinie est déplacée à une destination qui est dans la portée du même verrou (par exemple, dans l'arborescence d'espace de noms d'URL couverte par le verrou) la ressource déplacée va là encore être ajoutée au verrou. Dans ces deux exemples, comme spécifié au paragraphe 7.5, un en-tête If doit être soumis et contenir un jeton de verrou pour la source et pour la destination.

7.7. Rafraîchissement des verrous en écriture

Un client NE DOIT PAS soumettre deux fois la même demande de verrou en écriture. Noter qu'un client sait toujours qu'il resoumet la même demande de verrou parce que il doit inclure le jeton de verrou dans l'en-tête If afin de faire la demande pour une ressource qui est déjà verrouillée.

Cependant, un client peut soumettre une demande LOCK avec un en-tête If mais sans corps. Un serveur qui reçoit une demande LOCK sans corps NE DOIT PAS créer un nouveau verrou -- cette forme de la demande LOCK n'est à utiliser que pour "rafraîchir" un verrou existant (ce qui signifie, au minimum, que tous les temporisateurs associés au verrou DOIVENT être relancés).

Les clients peuvent soumettre les en-têtes Timeout de valeur arbitraire avec leurs demandes de rafraîchissement de verrou. Les serveurs, comme toujours, peuvent ignorer les en-têtes Timeout soumis par les client, et un serveur PEUT rafraîchir un verrou avec une période de temporisation qui est différente de la période de temporisation précédente utilisée pour le verrou, pourvu qu'il annonce la nouvelle valeur dans la réponse de rafraîchissement de LOCK.

Si une erreur est reçue dans la réponse à une demande de rafraîchissement de LOCK, le client NE DOIT PAS supposer que le verrou a été rafraîchi.

8. Traitement général des demandes et des réponses

8.1 Préséance dans le traitement d’erreurs

Les serveurs DOIVENT retourner des erreurs d'autorisation, de préférence aux autres erreurs. Cela évite des fuites d'informations sur des ressources protégées (par exemple, un client qui trouve qu'une ressource cachée existe en voyant une réponse 423 Verrouillé à une demande anonyme sur la ressource).

8.2 Utilisation de XML

Dans HTTP/1.1, les informations de paramètre de méthode étaient exclusivement codées dans les en-têtes HTTP. À la différence de HTTP/1.1, WebDAV code les informations de paramètre de méthode soit dans un corps d'entité de demande XML ([XML]) soit dans un en-tête HTTP. L'utilisation de XML pour coder les paramètres de méthode est motivée par la capacité d'ajouter des éléments XML supplémentaires aux structures existantes, assurant l'extensibilité; et par la capacité de XML à coder les informations dans les jeux de caractères ISO 10646, fournissant la prise en charge de l'internationalisation.

En plus des paramètres de méthode de codage, XML est utilisé dans WebDAV pour coder les réponses provenant des méthodes, fournissant les avantages d'extensibilité et d'internationalisation de XML pour les résultats de méthode, aussi bien que pour les entrées.

Quand XML est utilisé pour un corps de demande ou de réponse, le type de contenu DEVRAIT être application/xml. Les mises en œuvre DOIVENT accepter text/xml et application/xml dans les corps de demande et de réponse. L'utilisation de text/xml est déconseillée.

Tous les clients et ressources conformes à DAV DOIVENT utiliser des analyseurs XML conformes à [XML] et [XML-NAMES]. Tout XML utilisé dans les demandes ou réponses DOIT être, au minimum, bien formé et utiliser correctement les espaces de noms. Si un serveur reçoit du XML qui nest pas bien formé, il DOIT rejeter la demande entière avec un code 400 (Mauvaise demande). Si un client reçoit du XML qui n'est pas bien formé dans une réponse, alors le client NE DOIT PAS supposer quelque chose sur le résultat de la méthode exécutée et DEVRAIT traiter le serveur comme dysfonctionnant.

Noter que le traitement de XML soumis par une source qui n'est pas de confiance peut causer des risques pour la confidentialité, la sécurité, et la qualité de service (voir la Section 20). Les serveurs PEUVENT rejeter des demandes suspectes (même si elles consistent en XML bien formé) par exemple, avec un code d'état 400 (Mauvaise demande) et un corps de réponse facultatif expliquant le problème.

8.3 Traitement d'URL

Les URL apparaissent en de nombreux endroits dans les demandes et réponses. L'expérience de l'interopérabilité de la [RFC2518] a montré que de nombreux clients qui analysent des réponses multi états ne mettent pas pleinement en œuvre la résolution de référence définie à la Section 5 de la [RFC3986]. Donc, les serveurs doivent en particulier être vigilants lors du traitement des URL dans les réponses, pour s'assurer que les clients ont assez de contexte pour être capables d'interpréter tous les URL. Les règles de cette section s'appliquent non seulement aux URL de ressources dans l'élément 'href' dans les réponses multi états, mais aussi aux URL de ressource d'en-tête Destination et If.

L'envoyeur a le choix entre deux approches : utiliser une référence relative, qui est résolue contre l'URI de demande, ou un URI complet. Un serveur DOIT s'assurer que chaque valeur 'href' dans une réponse multi états utilise le même format.

WebDAV utilise seulement une forme de référence relative dans ses extensions, le chemin absolu.

Simple-ref = URI absolu | (chemin absolu ["?" interrogation])

Les productions URI absolu, chemin absolu et interrogation sont définies aux paragraphes 4.3, 3.3, et 3.4 de la [RFC3986].

Dans les productions Simple-ref, les envoyeurs NE DOIVENT PAS utiliser les segments de points ("." ou ".."). Les productions Simple-ref utilisées dans des réponses multi états NE DOIVENT PAS avoir de préfixes qui ne correspondent pas à l'URI de demande (en utilisant les règles de comparaison définies au paragraphe 3.2.3 de la [RFC2616]).

Les identifiants de collections DEVRAIENT se terminer par un caractère '/'.

8.3.1 Exemple – Traitement correct d'URL

Considérons la collection http://exemple.com/sample/ avec l'URI de membre interne http://exemple.com/sample/a%20test et la demande PROPFIND ci-dessous :

>>Demande :

```
PROPFIND /sample/ HTTP/1.1
  Host: exemple.com
  Depth: 1
```

Dans ce cas, le serveur devrait retourner deux éléments 'href' contenant soit
- o 'http://exemple.com/sample/' et 'http://exemple.com/sample/a%20test', soit
- o '/sample/' et '/sample/a%20test'

Noter que même si le serveur peut mémoriser la ressource membre en interne comme 'test', elle doit être codée en pourcentage quand elle est utilisée à l'intérieur d'une référence d'URI (voir au paragraphe 2.1 de la [RFC3986]). Noter aussi qu'un URI légal peut quand même contenir des caractères qui doivent être échappés dans les données de caractères XML, comme le caractère esperluettte *(&)*.

8.4 Corps requis dans les demandes

Certaines de ces nouvelles méthodes ne définissent pas de corps. Les serveurs DOIVENT examiner toutes les demandes pour chercher un corps, même quand un corps n'est pas attendu. Dans les cas où un corps de demande est présent mais va être ignoré par un serveur, le serveur DOIT rejeter la demande avec le code 415 (Type de support non pris en charge). Cela informe le client (qui peut avoit tenté d'utiliser une extension) que le corps ne pourra pas être traité comme le client le voulait.

8.5 En-têtes HTTP à utiliser dans WebDAV

HTTP définit de nombreux en-têtes qui peuvent être utilisés dans les demandes et réponses WebDAV. Tous ne sont pas appropriés dans toutes les situations et certaines interactions peuvent être indéfinies. Noter que HTTP 1.1 exige, si possible, l'en-tête Date dans toutes les réponses (voir au paragraphe 14.18 de la [RFC2616]).

Le serveur DOIT faire les vérifications d'autorisation avant de vérifier tout en-tête HTTP conditionnel.

8.6 ETag

HTTP 1.1 recommande l'utilisation des ETag plutôt que des dates de modification, pour le contrôle des antémémoires, et il y a des raisons encore plus fortes de préférer les ETag pour les informations d'auteur. L'utilisation correcte des ETag est encore plus importante dans un environnement d'informations d'auteur réparties, parce que les ETag sont nécessaires avec les verrous pour éviter le problème de la mise à jour perdue. Un client pourrait échouer à renouveler un verrou, par exemple, quand le verrou arrive en fin de temporisation et que le client est accidentellement hors ligne ou au milieu d'un long téléchargement. Quand un client échoue à renouveler le verrou, il est possible que la ressource puisse quand même être reverrouillée et que l'utilisateur puisse poursuivre l'édition, pour autant qu'aucun changement n'ait été fait entre temps. Les ETag sont exigées pour que le client soit capable de distinguer ce cas. Autrement, le client est forcé de demander à l'utilisateur si il doit écraser la ressource sur le serveur sans même être capable de dire à l'utilisateur si elle a changé. Les horodatages ne résolvent pas ce problème aussi bien que les ETag.

Les ETag fortes sont beaucoup plus utiles pour les cas d'utilisation d'informations d'auteurs que des ETag faibles (voir au paragraphe 13.3.3 de la [RFC2616]). L'équivalence sémantique peut être un concept utile mais qui dépend du type de document et du type d'application, et l'interopérabilité peut exiger un accord ou une norme qui sort du domaine d'application de la présente spécification et de HTTP. Noter aussi que les ETag faibles ont certaines restrictions dans HTTP, par exemple, elles ne peuvent pas être utilisées dans les en-têtes If-Match.

Noter que la signification d'une ETag dans une réponse PUT n'est pas clairement définie, ni dans le présent document ni dans la RFC 2616 (c'est-à-dire, si la ETag signifie que la ressource est équivalente octet pour octet au corps de la demande PUT, ou si le serveur pourrait avoir fait des changements mineurs dans le formatage ou le contenu du document lors de la mémorisation). Ceci est un problème de HTTP, pas vraiment un problème de WebDAV.

Parce que les clients peuvent être forcés d'interroger les utilisateurs ou d'éliminer un contenu changé si la ETag change, un serveur WebDAV NE DEVRAIT PAS changer une ETag (ou l'heure de dernière modification) pour une ressource qui a un corps et une localisation inchangés. La ETag représente l'état du corps ou du contenu de la ressource. Il n'y a pas de façon similaire de dire si les propriétés ont changé.

8.7 Inclusion de corps de réponse d'erreur

HTTP et WebDAV n'utilisaient pas le corps de la plupart des réponses d'erreur pour les informations analysables par la machine jusqu'à ce que la spécification des extensions d'informations de version à WebDAV introduise un mécanisme pour inclure des informations plus spécifiques dans le corps d'une réponse d'erreur (paragraphe 1.6 de la [RFC3253]). Le mécanisme de corps d'erreur est approprié avec toute réponse d'erreur qui peut prendre un corps mais n'en a pas déjà de

défini. Le mécanisme est particulièrement approprié quand un code d'état peut signifier de nombreuses choses (par exemple, "400 Mauvaise demande" peut signifier que des en-têtes exigés sont manquants, que des en-têtes ont un format incorrect, ou bien d'autres choses). Ce mécanisme de corps d'erreur est traité à la Section 16.

8.8 Impact des opérations d'espace de noms sur les valideurs d'antémémoire

Noter que les en-têtes de réponse HTTP "Etag" et "Last-Modified" (voir les paragraphes 14.19 et 14.29 de la [RFC2616]) sont définis par URL (non par ressource) et sont utilisés par les clients pour la mise en antémémoire. Donc les serveurs doivent s'assurer que l'exécution de toute opération qui affecte l'espace de noms d'URL (comme COPY, MOVE, DELETE, PUT, ou MKCOL) préserve bien sa sémantique, en particulier :

- pour tout URL, la valeur "Last-Modified" DOIT s'incrémenter chaque fois que la représentation retournée sur GET change (dans les limites de la résolution d'horodatage) ;
- pour tout URL, une valeur de "ETag" NE DOIT PAS être réutilisée pour différentes représentations retournées par GET.

En pratique, cela signifie que les serveurs

- peuvent avoir à incrémenter les horodatages de "Last-Modified" pour chaque ressource dans l'espace de noms de destination d'une opération d'espace de noms sauf si cela peut être fait de façon plus sélective, et
- de même, peuvent devoir réallouer les valeurs de "ETag" pour ces ressources (sauf si le serveur alloue des étiquettes d'entité d'une façon telle qu'elles soient uniques sur tout l'espace de noms d'URL géré par le serveur).

Noter que ces considérations s'appliquent aussi à des cas d'utilisation spécifiques, comme d'utiliser PUT pour créer une nouvelle ressource à un URL qui avait été transposé auparavant, mais a été supprimé depuis.

Finalement, les propriétés WebDAV (telles que DAV:getetag et DAV:getlastmodified) qui héritent leur sémantique d'en-têtes HTTP doivent se comporter en conséquence.

9. Méthodes HTTP pour les noms d'auteurs distribués

9.1 Méthode PROPFIND

La méthode PROPFIND restitue des propriétés définies sur la ressource identifiée par l'URI de demande, si la ressource n'a pas de membre interne, ou sur la ressource identifiée par l'URI de demande et potentiellement ses ressources membres, si la ressource est une collection qui a des URL membres internes. Toutes les ressources conformes à DAV DOIVENT prendre en charge la méthode PROPFIND et l'élément XML propfind (paragraphe 14.20) avec tous les éléments XML définis pour être utilisés avec cet élément.

Un client DOIT soumettre un en-tête Depth d'une valeur de "0", "1", ou "infini" avec une demande PROPFIND. Les serveurs DOIVENT prendre en charge les demandes de profondeur "0" et "1" sur les ressources conformes à WebDAV et DEVRAIENT prendre en charge les demandes "infini". En pratique, la prise en charge des demandes de profondeur infinie PEUT être désactivée, du fait des problèmes de performances et de sécurité associés à ce comportement. Les serveurs DEVRAIENT traiter une demande sans en-tête Depth comme si un en-tête "Depth: infinity" était inclus.

Un client peut soumettre un élément XML 'propfind' dans le corps de la méthode de demande qui décrit quelles informations sont demandées. Il est possible de :

- demander des valeurs de propriété particulières, en nommant les propriétés désirées dans l'élément 'prop' (l'ordre des propriétés PEUT être ignoré par le serveur),
- demander les valeurs de propriété pour les propriétés définies dans la présente spécification (au minimum) plus des propriétés mortes, en utilisant l'élément 'allprop' (l'élément 'include' peut être utilisé avec 'allprop' pour dire au serveur d'inclure aussi des propriétés vives qui ne pourraient pas avoir été retournées autrement),
- demander une liste des noms de toutes les propriétés définies sur la ressource, en utilisant l'élément 'propname'.

Un client peut choisir de ne pas soumettre de corps de demande. Un corps de demande PROPFIND vide DOIT être traité comme si c'était une demande 'allprop'.

Noter que 'allprop' ne retourne pas de valeurs pour toutes les propriétés vives. Les serveurs WebDAV ont de plus en plus des propriétés dont le calcul est coûteux ou qui sont longues (voir les [RFC3253] et [RFC3744]) et ne retournent pas toutes les propriétés. Les clients WebDAV peuvent plutôt utiliser les demandes propname pour découvrit quelles propriétés vives existent, et demander des propriétés désignées quand ils restituent les valeurs. Pour une propriété vive définie ailleurs, cette definition peut spécifier si cette propriété vive va être retournée dans les demandes 'allprop'.

Tous les serveurs DOIVENT prendre en charge le retour d'une réponse de type de contenu text/ xml ou application/xml qui

contient un élément XML multi états décrivant les résultats des tentatives de restituer les diverses propriétés.

Si il y a une erreur de restitution d'une propriété, un résultat d'erreur approprié DOIT être inclus dans la réponse. Une demande de restitution de la valeur d'une propriété qui n'existe pas est une erreur et DOIT être notée par un élément XML 'response' qui contient une valeur d'état 404 (Non trouvé).

Par conséquent, l'élément XML 'multistatus' pour une ressource de collection DOIT inclure un élément XML 'response' pour chaque URL membre de la collection, quelle que soit la profondeur demandée. Il NE DEVRAIT PAS inclure d'élément 'response' pour les ressources qui ne sont pas conformes à WebDAV. Chaque élément 'response' DOIT contenir un élément 'href' qui contient l'URL de la ressource sur laquelle les propriétés dans l'élément XML "prop" sont définies. Les résultats pour une demande PROPFIND sur une ressource de collection sont retournées comme une liste plate dont l'ordre des entrées n'est pas significatif. Noter qu'une ressource peut avoir seulement une valeur pour une propriété d'un nom donné, de sorte que la propriété peut seulement se montrer une fois dans les réponses PROPFIND.

Les propriétés peuvent être soumises à un contrôle d'accès. Dans le cas des demandes 'allprop' et 'propname', si un principal n'a pas le droit de savoir si une propriété particulière existe, alors la propriété PEUT être exclue en silence de la réponse.

Certains résultats PROPFIND PEUVENT être mis en antémémoire, avec des précautions, car il n'y a pas de mécanisme de validation d'antémémoire pour la plupart des propriétés. Cette méthode est sûre et idempotente (voir au paragraphe 9.1 de la [RFC2616]).

9.1.1 Codes d’état PROPFIND

Ce paragraphe, comme les paragraphes similaires pour les autres méthodes, donne des lignes directrices sur les codes d'erreur et les préconditions ou postconditions (définies à la Section 16) qui pourraient être particulièrement utiles avec PROPFIND.

403 Interdit - un serveur PEUT rejeter les demandes PROPFIND sur les collections avec un en-tête de profondeur "Infini", et dans ce cas, il DEVRAIT utiliser cette erreur avec le code de précondition 'propfind-finite-depth' dans le corps de l'erreur.

9.1.2 Codes d’état à utiliser dans l’élément 'propstat'

Dans les réponses PROPFIND, des informations sur les propriétés individuelles sont retournées dans les éléments 'propstat' (voir au paragraphe 14.22) chacun contenant un élément individuel 'status' contenant les informations sur les propriétés qui apparaissent dedans. La liste ci-dessous résume les codes d'état les plus courants utilisés dans 'propstat'; cependant, les clients devraient être prêts à traiter aussi d'autres codes des séries 2/3/4/5xx.

200 OK - une propriété existe et/ou sa valeur est bien retournée.

401 Non autorisé - la propriété ne peut pas être vue sans une autorisation appropriée.

403 Interdit - la propriété ne peut pas être vue quelle que soit l'authentification.

404 Non trouvée - la propriété n'existe pas.

9.1.3 Exemple – Restitution des propriétés nommées

>>Demande

```
PROPFIND /file HTTP/1.1
Host: www.exemple.com
Content-type: application/xml; charset="utf-8"
Content-Length: xxxx

<?xml version="1.0" encoding="utf-8" ?>
<D:propfind xmlns:D="DAV:">
  <D:prop xmlns:R="http://ns.exemple.com/boxschema/">
    <R:bigbox/>
    <R:author/>
    <R:DingALing/>
    <R:Random/>
  </D:prop>
```

```
</D:propfind>

>>Réponse

HTTP/1.1 207 Multi-Status
Content-Type: application/xml; charset="utf-8"
Content-Length: xxxx

<?xml version="1.0" encoding="utf-8" ?>
<D:multistatus xmlns:D="DAV:">
  <D:response xmlns:R="http://ns.exemple.com/boxschema/">
    <D:href>http://www.exemple.com/file</D:href>
    <D:propstat>
      <D:prop>
        <R:bigbox>
          <R:BoxType>Box type A</R:BoxType>
        </R:bigbox>
        <R:author>
          <R:Name>J.J. Johnson</R:Name>
        </R:author>
      </D:prop>
      <D:status>HTTP/1.1 200 OK</D:status>
    </D:propstat>
    <D:propstat>
      <D:prop><R:DingALing/><R:Random/></D:prop>
      <D:status>HTTP/1.1 403 Interdit</D:status>
      <D:responsedescription> L'usager n'a pas accès à la propriété DingALing.
      </D:responsedescription>
    </D:propstat>
  </D:response>
  <D:responsedescription> Il y a eu une erreur de violation d'accès.
  </D:responsedescription>
</D:multistatus>
```

Dans cet exemple, PROPFIND est exécuté sur une ressource non collection http://www.exemple.com/file. L'élément XML propfind spécifie le nom de quatre propriétés dont les valeurs sont demandées. Dans ce cas, seulement deux propriétés ont été retournées, car le principal faisant la demande n'a pas des droits d'accès suffisants pour voir les propriétés trois et quatre.

9.1.4 Exemple – Utilisation de 'propname' pour restituer tous les noms de propriété

```
>>Demande

PROPFIND /conteneur/ HTTP/1.1
Host: www.exemple.com
Content-Type: application/xml; charset="utf-8"
Content-Length: xxxx

<?xml version="1.0" encoding="utf-8" ?>
<propfind xmlns="DAV:">
  <propname/>
</propfind>

>>Réponse

HTTP/1.1 207 Multi-Status
Content-Type: application/xml; charset="utf-8"
Content-Length: xxxx

<?xml version="1.0" encoding="utf-8" ?>
<multistatus xmlns="DAV:">
  <response>
    <href>http://www.exemple.com/conteneur/</href>
```

```
   <propstat>
    <prop xmlns:R="http://ns.exemple.com/boxschema/">
     <R:bigbox/>
     <R:author/>
     <creationdate/>
     <displayname/>
     <ressourcetype/>
     <supportedlock/>
    </prop>
    <status>HTTP/1.1 200 OK</status>
   </propstat>
  </response>
  <response>
   <href>http://www.exemple.com/conteneur/front.html</href>
   <propstat>
    <prop xmlns:R="http://ns.exemple.com/boxschema/">
     <R:bigbox/>
     <creationdate/>
     <displayname/>
     <getcontentlength/>
     <getcontenttype/>
     <getetag/>
     <getlastmodified/>
     <ressourcetype/>
     <supportedlock/>
    </prop>
    <status>HTTP/1.1 200 OK</status>
   </propstat>
  </response>
</multistatus>
```

Dans cet exemple, PROPFIND est invoqué sur la ressource de collection http://www.exemple.com/conteneur/, avec un élément XML propfind contenant l'élément XML propname, signifiant que le nom de toutes les propriétés devrait être retourné. Comme aucun en-tête Depth n'est présent, on suppose sa valeur par défaut de "infini", signifiant que le nom des propriétés sur la collection et tous ses descendants devrait être retourné.

En cohérence avec l'exemple précédent, la ressource http://www.exemple.com/conteneur/ a six propriétés définies sur elle : bigbox et author dans l'espace de noms "http://ns.exemple.com/boxschema/", et creationdate, displayname, ressourcetype, et supportedlock dans l'espace de noms "DAV:".

La ressource http://www.exemple.com/conteneur/index.html, membre de la collection "conteneur", a neuf propriétés définies, bigbox dans l'espace de noms "http://ns.exemple.com/boxschema/" et creationdate, displayname, getcontentlength, getcontenttype, getetag, getlastmodified, ressourcetype, et supportedlock dans l'espace de noms "DAV:".

Cet exemple montre aussi l'utilisation de la portée d'espace de noms XML et de l'espace de noms par défaut. Comme l'attribut "xmlns" ne contient pas de préfixe, l'espace de noms s'applique par défaut à tous les éléments inclus. Donc, tous les éléments qui ne déclarent pas explicitement l'espace de noms auquel ils appartiennent sont membres de l'espace de noms "DAV:".

9.1.5 Exemple – Utilisation de ce qu'on appelle 'allprop'

Noter que 'allprop', en dépit de son nom, qui demeure pour la rétro-compatibilité, ne retourne pas toutes les propriété, mais seulement les propriétés mortes et les propriétés vives définies dans la présente spécification.

>>Demande

```
PROPFIND /conteneur/ HTTP/1.1
Host: www.exemple.com
Depth: 1
Content-Type: application/xml; charset="utf-8"
Content-Length: xxxx

<?xml version="1.0" encoding="utf-8" ?>
```

```
<D:propfind xmlns:D="DAV:">
  <D:allprop/>
</D:propfind>
```

>>Réponse

```
HTTP/1.1 207 Multi-Status
Content-Type: application/xml; charset="utf-8"
Content-Length: xxxx

<?xml version="1.0" encoding="utf-8" ?>
<D:multistatus xmlns:D="DAV:">
  <D:response>
    <D:href>/conteneur/</D:href>
    <D:propstat>
      <D:prop xmlns:R="http://ns.exemple.com/boxschema/">
        <R:bigbox><R:BoxType>Box type A</R:BoxType></R:bigbox>
        <R:author><R:Name>Hadrian</R:Name></R:author>
        <D:creationdate>1997-12-01T17:42:21-08:00</D:creationdate>
        <D:displayname>Exemple collection</D:displayname>
        <D:ressourcetype><D:collection/></D:ressourcetype>
        <D:supportedlock>
          <D:lockentry>
            <D:lockscope><D:exclusive/></D:lockscope>
            <D:locktype><D:write/></D:locktype>
          </D:lockentry>
          <D:lockentry>
            <D:lockscope><D:shared/></D:lockscope>
            <D:locktype><D:write/></D:locktype>
          </D:lockentry>
        </D:supportedlock>
      </D:prop>
      <D:status>HTTP/1.1 200 OK</D:status>
    </D:propstat>
  </D:response>
  <D:response>
    <D:href>/conteneur/front.html</D:href>
    <D:propstat>
      <D:prop xmlns:R="http://ns.exemple.com/boxschema/">
        <R:bigbox><R:BoxType>Box type B</R:BoxType>
        </R:bigbox>
        <D:creationdate>1997-12-01T18:27:21-08:00</D:creationdate>
        <D:displayname>Exemple HTML ressource</D:displayname>
        <D:getcontentlength>4525</D:getcontentlength>
        <D:getcontenttype>text/html</D:getcontenttype>
        <D:getetag>"zzyzx"</D:getetag>
        <D:getlastmodified
          >Mon, 12 Jan 1998 09:25:56 GMT</D:getlastmodified>
        <D:ressourcetype/>
        <D:supportedlock>
          <D:lockentry>
            <D:lockscope><D:exclusive/></D:lockscope>
            <D:locktype><D:write/></D:locktype>
          </D:lockentry>
          <D:lockentry>
            <D:lockscope><D:shared/></D:lockscope>
            <D:locktype><D:write/></D:locktype>
          </D:lockentry>
        </D:supportedlock>
      </D:prop>
      <D:status>HTTP/1.1 200 OK</D:status>
    </D:propstat>
  </D:response>
```

</D:multistatus>

Dans cet exemple, PROPFIND a été invoqué sur la ressource http://www.exemple.com/conteneur/ avec un en-tête Depth de 1, ce qui signifie que la demande s'applique à la ressource et ses enfants, et à un élément XML propfind contenant l'élément XML allprop, ce qui signifie que la demande devrait retourner le nom et la valeur de toutes les propriétés mortes définies sur les ressources, plus le nom et la valeur de toutes les propriétés définies dans la présente spécification. Cet exemple illustre l'utilisation de références relatives dans les éléments 'href' de la réponse.

La ressource http://www.exemple.com/conteneur/ a six propriétés définies sur elle : 'bigbox' et 'author' dans l'espace de noms "http://ns.exemple.com/boxschema/", DAV:creationdate, DAV: displayname, DAV:ressourcetype, et DAV:supportedlock.

Les quatre dernières propriétés sont spécifiques de WebDAV, définies à la Section 15. Comme GET n'est pas pris en charge sur cette ressource, les propriétés get* (par exemple, DAV:getcontentlength) ne sont pas définies sur cette ressource. Les propriétés spécifiques de WebDAV affirment que "conteneur" a été créé le 1er décembre 1997, à 5:42:21 PM, dans une zone horaire 8 heures à l'ouest de GMT (DAV:creationdate), a pour nom "Exemple collection" (DAV: displayname), un type de ressource de collection (DAV:ressourcetype), et prend en charge les verrous exclusifs et partagés en écriture (DAV:supportedlock).

La ressource http://www.exemple.com/conteneur/front.html a neuf propriétés définies sur elle :
'bigbox' dans l'espace de noms "http://ns.exemple.com/boxschema/" (une autre instance du type de propriété "bigbox") DAV:creationdate, DAV: displayname, DAV:getcontentlength, DAV:getcontenttype, DAV:getetag, DAV:getlastmodified, DAV:ressourcetype, et DAV:supportedlock.

Les propriétés spécifiques de DAV affirment que "front.html" a été créée le 1er décembre 1997, à 6:27:21 PM, dans une zone horaire 8 heures à l'ouest de GMT (DAV:creationdate) a un nom de "Exemple HTML ressource" (DAV: displayname), une longueur de contenu de 4525 octets (DAV:getcontentlength) un type MIME de "text/html" (DAV:getcontenttype) une étiquette d'entité de "zzyzx" (DAV:getetag) a été modifiée en dernier le lundi 12 janvier 1998, à 09:25:56 GMT (DAV:getlastmodified) a un type de ressource vide, ce qui signifie qu'elle n'est pas une collection (DAV:ressourcetype) et prend en charge les verrous exclusifs et partagés en écriture (DAV:supportedlock).

9.1.6 Exemple – Utilisation de 'allprop' avec 'include'

>>Demande

```
PROPFIND /mycol/ HTTP/1.1
Host: www.exemple.com
Depth: 1
Content-Type: application/xml; charset="utf-8"
Content-Length: xxxx

<?xml version="1.0" encoding="utf-8" ?>
<D:propfind xmlns:D="DAV:">
  <D:allprop/>
  <D:include>
    <D:supported-live-property-set/>
    <D:supported-report-set/>
  </D:include>
</D:propfind>
```

Dans cet exemple, PROPFIND est exécuté sur la ressource http://www.exemple.com/mycol/ et ses ressources membres internes. Le client demande les valeurs de toutes les propriétés vives définies dans la présente spécification, plus toutes les propriétés mortes, plus deux propriétés vives supplémentaires définies dans la [RFC3253]. La réponse n'est pas montrée.

9.2 Méthode PROPPATCH

La méthode PROPPATCH traite les instructions spécifiées dans le corps de la demande pour établir et/ou supprimer des propriétés définies sur la ressource identifiée par l'URI de demande.

Toutes les ressources conformes à DAV DOIVENT prendre en charge la méthode PROPPATCH et DOIVENT traiter les instructions qui sont spécifiées en utilisant les éléments XML "propertyupdate", "set", et "remove". L'exécution des directives dans cette méthode est, bien sûr, soumise aux contraintes de contrôle d'accès. Les ressources conformes à DAV

DEVRAIENT prendre en charge l'établissement de propriétés mortes arbitraires.

Le corps du message de demande d'une méthode PROPPATCH DOIT contenir l'élément XML "propertyupdate".

Les serveurs DOIVENT traiter les éléments fils de l'élément XML "propertyupdate" dans l'ordre où ils apparaissent dans le corps de la demande (exception à la règle normale que l'ordre n'est pas significatif). Les instructions DOIVENT être exécutées soit toutes, soit aucune. Donc, si une erreur se produit durant le traitement, toutes les instructions exécutées DOIVENT être défaites et un résultat d'erreur approprié retourné. Les détails du traitement des instructions se trouvent dans la définition des instructions set et remove aux paragraphes 14.23 et 14.26.

Si un serveur tente de faire des changements de propriété dans une demande PROPPATCH (c'est-à-dire, si la demande n'est pas rejetée pour des erreurs de haut niveau avant de traiter le corps) la réponse DOIT être multi états comme décrit au paragraphe 9.2.1.

Cette méthode est idempotente, mais non sûre (voir au paragraphe 9.1 de la [RFC2616]). Les réponses à cette méthode NE DOIVENT PAS être mises en antémémoire.

9.2.1 Codes d'état à utiliser dans l'élément 'propstat'

Dans les réponses PROPPATCH, les informations sur les propriétés individuelles sont retournées dans les éléments 'propstat' (voir au paragraphe 14.22) chacun contenant un élément 'status' individuel contenant les informations sur les propriétés qui y apparaissent. La liste ci-dessous résume les codes d'état les plus courants utilisés dans 'propstat' ; cependant, les clients devraient être prêts à traiter aussi d'autres codes des séries 2/3/4/5xx.

200 (OK) - l'établissement ou changment de la propriété a réussi. Noter que si cela apparaît pour une propriété, cela apparaît pour toutes les propriétés de la réponse, du fait de l'atomicité de PROPPATCH.

403 (Interdit) - le client, pour des raisons que le serveur choisit de ne pas spécifier, ne peut pas altérer une des propriétés.

403 (Interdit) - le client a tenté d'établir une propriété protégée, comme DAV:getetag. Si il retourne cette erreur, le serveur DEVRAIT utiliser le code de précondition "ne-peut-pas-modifier-une-propriété-protégée" dans le corps de réponse.

409 (Conflit) - le client a fourni une valeur dont la sémantique n'est pas appropriée pour la propriété.

424 (Échec de dépendance) - le changement de propriété ne peut pas être fait parce que un autre changement de propriété a échoué.

507 (Mémoire insuffisante) - le serveur n'a pas l'espace suffisant pour enregistrer la propriété.

9.2.2 Exemple - PROPPATCH

>>Demande

```
PROPPATCH /bar.html HTTP/1.1
Host: www.exemple.com
Content-Type: application/xml; charset="utf-8"
Content-Length: xxxx

<?xml version="1.0" encoding="utf-8" ?>
<D:propertyupdate xmlns:D="DAV:"
       xmlns:Z="http://ns.exemple.com/standards/z39.50/">
  <D:set>
    <D:prop>
      <Z:Authors>
        <Z:Author>Jim Whitehead</Z:Author>
        <Z:Author>Roy Fielding</Z:Author>
      </Z:Authors>
    </D:prop>
  </D:set>
  <D:remove>
    <D:prop><Z:Copyright-Owner/></D:prop>
  </D:remove>
</D:propertyupdate>
```

>>Réponse

```
HTTP/1.1 207 Multi-Status
Content-Type: application/xml; charset="utf-8"
Content-Length: xxxx

<?xml version="1.0" encoding="utf-8" ?>
<D:multistatus xmlns:D="DAV:"
      xmlns:Z="http://ns.exemple.com/standards/z39.50/">
  <D:response>
    <D:href>http://www.exemple.com/bar.html</D:href>
    <D:propstat>
      <D:prop><Z:Authors/></D:prop>
      <D:status>HTTP/1.1 424 Failed Dependency</D:status>
    </D:propstat>
    <D:propstat>
      <D:prop><Z:Copyright-Owner/></D:prop>
      <D:status>HTTP/1.1 409 Conflict</D:status>
    </D:propstat>
    <D:responsedescription> Le propriétaire du copyright ne peut pas être supprimé ou altéré.</D:responsedescription>
  </D:response>
</D:multistatus>
```

Dans cet exemple, le client demande au serveur de régler la valeur de la propriété "Authors" dans l'espace de noms "http://ns.exemple.com/standards/z39.50/", et de supprimer la propriété "Copyright-Owner" dans le même espace de noms. Comme la propriété Copyright-Owner ne peut pas être supprimée, aucune modification de propriété ne se produit. Le code d'état 424 (Échec de dépendance) pour la propriété "Authors" indique que cette action aurait réussi si il n'y avait pas le conflit avec la suppression de la propriété Copyright-Owner.

9.3 Méthode MKCOL

La méthode MKCOL est utilisée pour créer une nouvelle collection. Toutes les ressources conformes à DAV DOIVENT prendre en charge la méthode MKCOL.

MKCOL crée une nouvelle ressource de collection à la localisation spécifiée par l'URI de demande. Si l'URI de demande est déjà transposé en ressource, le MKCOL DOIT échouer. Durant le traitement de MKCOL, un serveur DOIT faire de l'URI de demande un membre interne de sa collection parente, sauf si l'URI de demande est "/". Si il n'existe pas un tel ancêtre, la méthode DOIT échouer. Quand l'opération MKCOL crée une nouvelle ressource de collection, tous les ancêtres DOIVENT déjà exister, ou la méthode DOIT échouer avec un code d'état 409 (Conflit). Par exemple, si une demande de créer une collection /a/b/c/d/ est faite, et si /a/b/c/ n'existe pas, la demande doit échouer.

Quand MKCOL est invoqué sans corps de demande, la nouvelle collection créée DEVRAIT n'avoir aucun membre.

Un message de demande MKCOL peut contenir un corps de message. Le comportement précis d'une demande MKCOL quand le corps est présent est indéfini, mais limité à la création de collections, membres d'une collection, corps de membres, et propriétés sur les collections ou membres. Si le serveur reçoit un type d'entité de demande MKCOL qu'il ne comprend pas ou ne prend pas en charge, il DOIT répondre par un code d'état 415 (Type de support non pris en charge). Si le serveur décide de rejeter la demande sur la base de la présence d'une entité ou du type d'une entité, il devrait utiliser le code d'état 415 (Type de support non pris en charge).

Cette méthode est idempotente, mais non sûre (voir au paragraphe 9.1 de la [RFC2616]). Les réponses à cette méthode NE DOIVENT PAS être mises en antémémoire.

9.3.1 Codes d'état MKCOL

En plus des codes d'état généraux possibles, les codes d'état suivants ont une applicabilité spécifique pour MKCOL :

201 (Créée) - la collection a été créée.

403 (Interdit) - cela indique au moins une des deux conditions :

1) le serveur ne permet pas la création de collections à la localisation données dans son espéce de noms d'URL, ou

2) la collection parente de l'URI de demande existe mais ne peut pas accepter de membres.

405 (Méthode non permise) - MKCOL peut seulement être exécuté sur un URL non transposé.

409 (Conflit) - une collection ne peut pas être faite à l'URI de demande avant qu'une ou plusieurs collections intermédiaires aient été créées. Le serveur NE DOIT PAS créer automatiquement ces collections intermédiaires.

415 (Type de support non pris en charge) - le serveur ne prend pas en charge le type de corps de demande (bien que les corps soient légaux sur les demandes MKCOL, car la présente spécification n'en définit aucune, le serveur ne va probablement prendre en charge aucun type de corps).

507 (Mémoire insuffisante) - la ressource n'a pas l'espace suffisant pour enregistrer l'état de la ressource après l'exécution de cette méthode.

9.3.2 Exemple - MKCOL

Cet exemple crée une collection appelée /webdisc/xfiles/ sur le serveur www.exemple.com.

>>Demande

MKCOL /webdisc/xfiles/ HTTP/1.1
Host: www.exemple.com

>>Réponse

HTTP/1.1 201 Créée

9.4 GET, HEAD pour les collections

La sémantique de GET est inchangée quand elle est appliquée à une collection, car GET est défini comme "restituer toute information (sous la forme d'une entité) identifiée par l'URI de demande" [RFC2616]. GET, quand appliqué à une collection, peut retourner le contenu d'une ressource "index.html", une vue lisible par l'homme du contenu de la collection, ou quelque chose d'autre aussi. Donc, il est possible que le résultar d'un GET sur une collection n'ait pas de corrélation avec l'apparenance à la collection.

De même, comme la définition de HEAD est un GET sans corps de message de réponse, la sémantique de HEAD n'est pas modifiée quand elle est appliquée à des ressources de collection.

9.5 Méthode POST pour les collections

Comme par définition la fonction réelle effectuée par POST est déterminée par le serveur et dépend souvent de la ressource particulière, le comportement de POST appliqué aux collections ne peut pas être significativement modifié parce que il est largement indéfini. Donc, la sémantique de POST n'est pas modifiée quand elle s'applique à une collection.

9.6 Exigences pour DELETE

DELETE est défini au paragraphe 9.7 de la [RFC2616] comme "supprime la ressource identifiée par l'URI de demande". Cependant, WebDAV change certaines exigences de traitement des DELETE.

Un serveur qui traite une demande DELETE réussie :
DOIT détruire les verrous dont la racine est sur la ressource supprimée
DOIT supprimer la transposition de l'URI de demande en toute ressource.

Donc, après une opération DELETE réussie (et en l'absence d'autres actions) une demande GET/HEAD/PROPFIND suivante à l'URI de demande cible DOIT retourner un code d'état 404 (Non trouvée).

9.6.1 DELETE pour les collections

La méthode DELETE sur une collection DOIT agir comme si un en-tête "Depth: infinity" était utilisé sur elle. Un client NE DOIT PAS soumettre un en-tête Depth avec un DELETE sur une collection avec une valeur autre que infini.

DELETE donne pour instruction que la collection spécifiée dans l'URI de demande et toutes les ressources identifiées par ses URL de membres internes soient supprimées.

Si une ressource identifiée par un URL de membre ne peut pas être supprimée, alors tous les ancêtres du membre NE DOIVENT PAS être supprimés, afin de conserver la cohérence de l'espace de noms d'URL.

Tous les en-têtes inclus avec DELETE DOIVENT être appliqués dans le traitement de chaque ressource à supprimer.

Quand la méthode DELETE a fini d'être traitée, il DOIT en résulter un espace de noms d'URL cohérent.

Si une erreur se produit dans la suppression d'une ressource membre (une ressource autre que la ressource identifiée dans l'URI de demande) alors la réponse peut être un 207 (Multi-états). Multi-états est utilisé ici pour indiquer quelles ressources internes ne pourraient PAS être supprimées, incluant un code d'erreur, qui devrait aider le client à comprendre quelles ressources ont causé la défaillance. Par exemple, le corps multi-états pourrait inclure une réponse avec l'état 423 (Verrouillé) si une ressource interne était verrouillée.

Le serveur PEUT retourner une réponse d'état 4xx, plutôt que 207, si la demande a complètement échoué.

Des codes d'état 424 (Échec de dépendance) NE DEVRAIENT PAS être dans la réponse 207 (Multi-états) pour DELETE. Ils peuvent être en toute sécurité laissés de côté parce que le client va savoir que les ancêtres d'une ressource ne pourraient pas être supprimés quand le client reçoit une erreur pour la progéniture de l'ancêtre. De plus, des erreurs 204 (Pas de contenu) NE DEVRAIENT PAS être retournées dans le 207 (Multi-états). La raison de cette interdiction est que 204 (Pas de contenu) est le code de succès par défaut.

9.6.2 Exemple - DELETE

>>Demande

```
DELETE /conteneur/ HTTP/1.1
Host: www.exemple.com
```

>>Réponse

```
HTTP/1.1 207 Multi-Status
Content-Type: application/xml; charset="utf-8"
Content-Length: xxxx

<?xml version="1.0" encoding="utf-8" ?>
<d:multistatus xmlns:d="DAV:">
  <d:response>
    <d:href>http://www.exemple.com/conteneur/ressource3</d:href>
    <d:status>HTTP/1.1 423 Locked</d:status>
    <d:error><d:lock-token-submitted/></d:error>
  </d:response>
</d:multistatus>
```

Dans cet exemple, la tentative de suppression de http://www.exemple.com/conteneur/ressource3 a échoué parce que elle est verrouillée, et aucun jeton de verrou n'a été soumis avec la demande. Par conséquent, la tentative de suppression de http://www.exemple.com/conteneur/ a aussi échoué. Donc, le client sait que la tentative de suppression de http://www.exemple.com/conteneur/ doit aussi avoir échoué car le parent ne peut pas être supprimé si son enfant ne l'a pas aussi été. Même si un en-tête Depth n'a pas été inclus, une profondeur de infini est supposée parce que la méthode est sur une collection.

9.7 Exigences pour PUT

9.7.1 PUT pour des ressources qui ne sont pas de collection

Un PUT effectué sur une ressource existante remplace l'entité de réponse GET de la ressource. Les propriétés définies sur la ressource peuvent être recalculées durant le traitement de PUT mais elles ne sont pas autrement affectées. Par exemple, si un serveur reconnaît le type de contenu du corps de la demande, il peut être capable d'extraire automatiquement les informations qui pourraient être avantageusement exposées comme propriétés.

Un PUT qui résulterait en la création d'une ressource sans une collection parente de portée appropriée DOIT échouer avec

un 409 (Conflit).

Une demande PUT permet à un client d'indiquer quel type de support a un corps d'entité, et si il devrait changer en cas d'écrasement. Donc, un client DEVRAIT fournir un Content-Type pour une nouvelle ressource si il en connaît une. Si le client ne fournit pas de Content-Type pour une nouvelle ressource, le serveur PEUT créer une ressource sans Content-Type alloué, ou il PEUT tenter d'allouer un Content-Type.

Noter que bien qu'un receveur devrait généralement traiter les métadonnées fournies avec une demande HTTP comme d'autorité, en pratique il n'est pas garanti qu'un serveur accepte les métadonnées fournies par un client (par exemple, tout en-tête de demande commençant par "Content-"). De nombreux serveurs ne permettent pas de configurer le Content-Type sur la base de la ressource en premier lieu. Donc, les clients ne peuvent pas toujours s'appuyer sur la capacité d'influencer directement le type de contenu en incluant un en-tête de demande Content-Type.

9.7.2 PUT pour les collections

La présente spécification ne définit pas le comportement de la méthode PUT pour les collections existantes. Une demande PUT sur une collection existante PEUT être traitée comme une erreur (405 Méthode non admise).

La méthode MKCOL est définie pour créer des collections.

9.8 Méthode COPY

La méthode COPY crée un duplicata de la ressource source identifiée par l'URI de demande, dans la ressource de destination identifiée par l'URI dans l'en-tête Destination. L'en-tête Destination DOIT être présent. Le comportement exact de la méthode COPY dépend du type de la ressource de source.

Toutes les ressources conformes à WebDAV DOIVENT prendre en charge la méthode COPY. Cependant, la prise en charge de la méthode COPY ne garantit pas la capacité de copier une ressource. Par exemple, des programmes séparés peuvent contrôler les ressources sur le même serveur. Par suite, il peut n'être pas possible de copier une ressource à une localisation qui paraît être sur le même serveur.

Cette méthode est idempotente, mais non sûre (voir au paragraphe 9.1 de la [RFC2616]). Les réponses à cette méthode NE DOIVENT PAS être mises en antémémoire.

9.8.1 COPY pour des ressources qui ne sont pas de collection

Quand la ressource de source n'est pas une collection, le résultat de la méthode COPY est la création d'une nouvelle ressource à la destination dont l'état et le comportement correspondent aussi étroitement que possible à celui de la ressource de source. Comme l'environnement à la destination peut être différent de celui de la source à cause de facteurs qui sortent de la portée du contrôle du serveur, comme l'absence des ressources nécessaires pour un fonctionnement correct, il peut n'être pas possible de dupliquer complètement le comportement de la ressource à la destination. Des altérations ultérieures à la ressource de destination ne vont pas modifier la ressource de source. Des altérations ultérieures à la ressource de source ne vont pas modifier la ressource de destination.

9.8.2 COPY pour les propriétés

Après une invocation réussie de COPY, toutes les propriétés mortes sur la ressource de source DEVRAIENT être dupliquées sur la ressource de destination. Les propriétés vives décrites dans le présent document DEVRAIENT être dupliquées comme des propriétés vives se comportant de façon identique à la ressource de destination, mais pas nécessairement avec les mêmes valeurs. Les serveurs NE DEVRAIENT PAS convertir les propriétés vives en propriétés mortes sur la ressource de destination, parce que les clients pourraient alors tirer des conclusions incorrectes sur l'état ou la fonction d'une ressource. Noter que certaines propriétés vives qui sont définies comme l'absence de la propriété ont une signification spécifique (par exemple, un fanion avec une signification "si il est présent", et l'opposé "si il est absent") et dans ces cas, une COPY réussie pourrait résulter en ce que la propriété soit rapportée comme "Non trouvée" dans les demandes suivantes.

Quand la destination est un URL non transposé, une opération COPY crée une nouvelle ressource un peu comme le fait une opération PUT. Les propriétés vives qui se rapportent à la création de ressource (comme DAV:creationdate) devraient avoir leurs valeurs réglées en conséquence.

9.8.3 COPY pour les collections

La méthode COPY sur une collection sans en-tête Depth DOIT agir comme si un en-tête Depth avec la valeur "infini" était

inclus. Un client peut soumettre un en-tête Depth sur un COPY sur une collection avec la valeur de "0" ou "infini". Les serveurs DOIT prendre en charge les comportements d'en-tête de profondeur "0" et "infini" sur les ressources conformes à WebDAV.

Un COPY de profondeur infinie donne pour instruction que la ressource de collection identifiée par l'URI de demande soit copiée à l'endroit identifié par l'URI dans l'en-tête Destination, et toutes ses ressources membres internes sont à copier à une localisation par rapport à lui, de façon récurrente à tous les niveaux de la hiérarchie de la collection. Noter qu'une profondeur infinie de COPY de /A/ dans /A/B/ pourrait conduire à une récurrence infinie si elle n'est pas traitée correctement.

Un COPY de "Depth: 0" donne seulement pour instruction que la collection et ses propriétés, mais pas les ressources identifiées par ses URL membres internes, sont à copier.

Tous les en-têtes inclus avec un COPY DOIVENT être appliqués dans le traitement de chaque ressource à copier, à l'exception de l'en-tête Destination.

L'en-tête Destination spécifie seulement l'URI de destination pour l'URI de demande. Quand elle est appliquée aux membres de la collection identifiée par l'URI de demande, la valeur de Destination est à modifier pour refléter la localisation actuelle dans la hiérarchie. Donc, si l'URI de demande est /a/ avec la valeur d'en-tête Host http://exemple.com/ et si Destination est http://exemple.com/b/, alors quand http://exemple.com/a/c/d est traité, il doit utiliser la destination de http://exemple.com/b/c/d.

Quand la méthode COPY a terminé son traitement, elle DOIT avoir créé un espace de noms d'URL cohérent à la destination (voir au paragraphe 5.1 la définition d'espace de noms cohérent). Cependant, si une erreur survient lors de la copie d'une collection interne, le serveur NE DOIT PAS copier de ressources identifiées par les membres de cette collection (c'est-à-dire, le serveur doit sauter cette sous arborescence) car cela créerait un espace de noms incohérent. Après la détection d'une erreur, l'opération COPY DEVRAIT essayer de finir autant de l'opération originale de copie que possible (c'est-à-dire, le serveur devrait quand même tenter de copier les autres sous arborescences et leurs membres qui ne sont pas des descendants de la collection qui cause l'erreur).

Donc, par exemple, si une opération de copie de profondeur infinie est effectuée sur la collection /a/, qui contient les collections /a/b/ et /a/c/, et si une erreur survient dans la copie de /a/b/, on devrait quand même tenter de copier /a/c/. De même, après avoir rencontré une erreur dans la copie d'une ressource non de collection au titre d'une copie de profondeur infinie, le serveur DEVRAIT essayer de finir autant de l'opération de copie originale que possible.

Si une erreur dans l'exécution de la méthode COPY se produit avec une ressource autre que celle identifiée dans l'URI de demande, la réponse DOIT alors être un 207 (Multi-états) et l'URL de la ressource qui cause la défaillance DOIT apparaître avec l'erreur spécifique.

Le code d'état 424 (Échec de dépendance) NE DEVRAIT PAS être retourné dans la réponse 207 (Multi-états) à partir d'une méthode COPY. Ces réponses peuvent être omises en toute sécurité parce que le client va savoir que la progéniture d'une ressource ne pourrait pas être copiée quand le client reçoit une erreur pour le parent. De plus, les codes d'état 201 (Créée) / 204 (Pas de contenu) NE DEVRAIENT PAS être retournés comme valeurs dans les réponses 207 (Multi-états) à des méthodes COPY. Ils peuvent eux aussi être omis en toute sécurité parce que ils sont les codes de succès par défaut.

9.8.4 COPY et écrasement d'une ressource de destination

Si une demande COPY a un en-tête Overwrite d'une valeur de "F", et si une ressource existe à l'URL de destination, le serveur DOIT faire échouer la demande.

Quand un serveur exécute une demande COPY et écrase une ressource de destination, le comportement exact PEUT dépendre de nombreux facteurs, incluant des capacités d'extension WebDAV (voir en particulier la [RFC3253]). Par exemple, quand une ressource ordinaire est écrasée, le serveur pourrait supprimer la ressource cible avant de faire la copie, ou pourrait faire un écrasement en place pour préserver les propriétés vives.

Quand une collection est écrasée, les membres de la collection de destination après la réussite de la demande COPY DOIVENT être les mêmes que ceux de la collection source immédiatement avant le COPY. Donc, fusionner les membres des collections de source et de destination dans la destination n'est pas un comportement conforme.

En général, si les clients demandent que l'état de l'URL de destination soit effacé avant un COPY (par exemple, pour forcer la réinitialisation des propriétés vives) alors le client pourrait envoyer un DELETE à la destination avant la demande COPY pour assurer cette réinitialisation.

9.8.5 Codes d'état

En plus des codes d'état généraux possibles, les codes d'état suivants ont une applicabilité spécifique pour COPY :

201 (Créée) - la ressource source a bien été copiée. L'opération COPY a résulté en la création d'une nouvelle ressource.

204 (Pas de contenu) - la ressource source a bien été copiée dans une ressource de destination préexistante.

207 (Multi-états) - plusieurs ressources devaient être affectées par le COPY, mais des erreurs sur certaines d'entre elles ont empêché l'opération d'avoir lieu. Des messages d'erreur spécifiques, ainsi que les URL les plus appropriés de source et destination, apparaissent dans le corps de la réponse multi-états. Par exemple, si une ressource de destination était verrouillée et n'a pas pu être écrasée, alors l'URL de ressource de destination apparaît avec l'état 423 (Verrouillé).

403 (Interdit) - l'opération est interdite. Un cas particulier pour COPY pourrait être que les ressources de source et destination soient la même ressource.

404 (Non trouvé) indique que le serveur d'origine n'a pas trouvé de représentation actuelle pour la ressource cible ou qu'il ne veut pas divulguer qu'il en existe une. Un code d'état 404 n'indique pas si ce manque de représentation est temporaire ou permanent ; le code d'état 410 (Parti) est préféré à 404 si le serveur d'origine sait, probablement par des moyens configurables, que la condition va être permanente.

409 (Conflit) - une ressource ne peut pas être créée à la destination tant qu'une ou plusieurs collections intermédiaires n'ont pas été créées. Le serveur NE DOIT PAS créer automatiquement ces collections intermédiaires.

412 (Échec de précondition) - une vérification d'en-tête de précondition a échoué, par exemple, l'en-tête Overwrite est "F" et l'URL de destination est déjà transposé en une ressource.

423 (Verrouillé) - la ressource de destination, ou ressource au sein de la collection de destination, était verrouillée. Cette réponse DEVRAIT contenir l'élément de précondition 'lock-token-submitted' *(jeton de verrouillage soumis)*.

502 (Mauvaise passerelle) - cela peut survenir quand la destination est sur un autre serveur, répertoire, ou espace de noms d'URL. Soit l'espace de noms de source ne prend pas en charge la copie dans l'espace de noms de destination, soit l'espace de noms de destination refuse d'accepter la ressource. Le client peut souhaiter essayer à la place GET/PUT et PROPFIND/PROPPATCH.

507 (Mémorisation insuffisante) - la ressource de destination n'a pas un espace suffisant pour enregistrer l'état de la ressource après l'exécution de cette méthode.

9.8.6 Exemple - COPY avec Overwrite

Cet exemple montre la ressource http://www.exemple.com/~fielding/index.html qui est copiée à la localisation http://www.exemple.com/users/f/fielding/index.html. Le code d'état 204 (Pas de contenu) indique que la ressource existante a été écrasée à la destination.

>>Demande

```
COPY /~fielding/index.html HTTP/1.1
Host: www.exemple.com
Destination: http://www.exemple.com/users/f/fielding/index.html
```

>>Réponse

```
HTTP/1.1 204 Pas de contenu
```

9.8.7 Exemple - COPY sans Overwrite

L'exemple suivant montre la même opération de copie, mais avec l'en-tête Overwrite réglé à "F". Une réponse de 412 (Échec de précondition) est retournée parce que l'URL de destination est déjà transposé en une ressource.

>>Demande

```
COPY /~fielding/index.html HTTP/1.1
Host: www.exemple.com
Destination: http://www.exemple.com/users/f/fielding/index.html
```

Overwrite: F

>>Réponse

HTTP/1.1 412 Échec de précondition

9.8.8 Exemple - COPY d'une collection

>>Demande

```
COPY /conteneur/ HTTP/1.1
Host: www.exemple.com
Destination: http://www.exemple.com/autreconteneur/
Depth: infini
```

>>Réponse

```
HTTP/1.1 207 Multi-Status
Content-Type: application/xml; charset="utf-8"
Content-Length: xxxx

<?xml version="1.0" encoding="utf-8" ?>
<d:multistatus xmlns:d="DAV:">
  <d:response>
    <d:href>http://www.exemple.com/autreconteneur/R2/</d:href>
    <d:status>HTTP/1.1 423 Locked</d:status>
    <d:error><d:lock-token-submitted/></d:error>
  </d:response>
</d:multistatus>
```

L'en-tête Depth est inutile car le comportement par défaut de COPY sur une collection est d'agir comme si un en-tête "Depth: infini" avait été soumis. Dans cet exemple, la plupart des ressources, ainsi que la collection, ont été copiées avec succès. Cependant, la collection R2 a échoué parce que la destination R2 est verrouillée. Parce qu'il y a eu une erreur pour la copie de R2, aucun des membres de R2 n'a été copié. Cependant, aucune erreur n'est mentionnée pour ces membres du fait des règles de minimisation des erreurs.

9.9 Méthode MOVE

L'opération MOVE sur une ressource non de collection est l'équivalent logique d'une copie (COPY) suivie par le traitement de maintenance de cohérence, suivi par une suppression de la source, où toutes les trois actions sont effectuées dans une seule opération. L'étape de maintenance de cohérence permet au serveur d'effectuer les mises à jour causées par le déplacement, comme de mettre à jour tous les URL autres que l'URI de demande qui identifie la ressource source, pour pointer sur la nouvelle ressource de destination.

L'en-tête Destination DOIT être présent sur toutes les méthodes MOVE et DOIT suivre toutes les exigences de COPY pour la partie COPY de la méthode MOVE. Toutes les ressources conformes à WebDAV DOIVENT prendre en charge la méthode MOVE.

La prise en charge de la méthode MOVE ne garantit pas la capacité de déplacer une ressource à une destination particulière. Par exemple, des programmes séparés peuvent en fait contrôler des ensembles différents de ressources sur lé même serveur. Donc, il peut n'être pas possible de déplacer une ressource au sein d'un espace de noms qui paraît appartenir au même serveur.

Si une ressource existe à la destination, la ressource de destination va être supprimée par un effet colatéral de l'opération MOVE, sous réserve des restrictions de l'en-tête Overwrite.

Cette méthode est idempotente, mais pas sûre (voir au paragraphe 9.1 de la [RFC2616]). Les réponses à cette méthode NE DOIVENT PAS être mises en antémémoire.

9.9.1 MOVE pour des propriétés

Les propriétés vives décrites dans le présent document DEVRAIENT être déplacées avec la ressource, afin que la ressource

ait les propriétés vives se comportant de façon identique à la ressource de destination, mais pas nécessairement avec les mêmes valeurs. Noter que certaines propriétés vives sont définies de façon telle que l'absence de la propriété ait une signification spécifique (par exemple, un fanion avec une signification "si il est présent", et le contraire "si il est absent") et dans ce cas, un MOVE réussi pourrait résulter en ce que la propriété soit rapportée comme "Non trouvée" dans les demandes suivantes. Si les propriétés vives ne vont pas fonctionner de la même façon à la destination, le serveur PEUT faire échouer la demande.

MOVE est fréquemment utilisé par les clients pour renommer un fichier sans changer sa collection parente, de sorte qu'il n'est pas approprié de réinitialiser toutes les propriétés vives qui sont établies à la création de la ressource. Par exemple, la valeur de propriété DAV:creationdate DEVRAIT rester la même après un MOVE.

Les propriétés mortes DOIVENT être déplacées avec la ressource.

9.9.2 MOVE pour des collections

Un MOVE avec "Depth: infini" donne pour instruction que la collection identifiée par l'URI de demande soit déplacée à l'adresse spécifiée dans l'en-tête Destination, et toutes les ressources identifiées par ses URL de membres internes sont à déplacer aux localisations qui s'y rapportent, de façon récurrente à travers tous les niveaux de la hiérarchie de collection.

La méthode MOVE sur une collection DOIT agir comme si un en-tête "Depth: infini" était utilisé sur elle. Un client NE DOIT PAS soumettre un en-tête Depth sur un MOVE sur une collection avec toute autre valeur que "infini".

Tous les en-têtes inclus avec MOVE DOIVENT être appliqués dans le traitement de chaque ressource à déplacer à l'exception de l'en-tête Destination. Le comportement de l'en-tête Destination est le même que pour COPY sur les collections.

Quand la méthode MOVE a achevé son traitement, elle DOIT avoir créé un espace de noms d'URL cohérent à la source et à la destination (voir au paragraphe 5.1 la définition de la cohérence d'espace de noms). Cependant, si une erreur se produit lors du déplacement d'une collection interne, le serveur NE DOIT PAS déplacer de ressources identifiées par les membres de la collection défaillante (c'est-à-dire, le serveur doit sauter la sous arborescence qui cause l'erreur) car cela créerait un espace de noms incohérent. Dans ce cas, après la détection de l'erreur, l'opération de déplacement DEVRAIT essayer de finir autant du déplacement original que possible (c'est-à-dire, le serveur devrait quand même tenter de déplacer les autres sous arborescences et les ressources identifiées par leurs membres qui ne sont pas des descendants d'une collection cause de l'erreur). Donc, par exemple, si un déplacement de profondeur infinie est effectué sur une collection /a/, qui contient les collections /a/b/ et /a/c/, et si une erreur se produit en déplaçant /a/b/, on devrait quand même tenter d'essayer de déplacer /a/c/. De même, après avoir rencontré une erreur pour le déplacement d'une ressource non de collection au titre d'un déplacement de profondeur infinie, le serveur DEVRAIT essayer de finir autant de l'opération de déplacement originale que possible.

Si une erreur se produit avec une ressource autre que la ressource identifiée dans l'URI de demande, la réponse DOIT alors être un 207 (Multi-états) et l'URL de la ressource erronée DOIT apparaître avec l'erreur spécifique.

Le code d'état 424 (Échec de dépendance) NE DEVRAIT PAS être retourné dans la réponse 207 (Multi-états) pour une méthode MOVE. Ces erreurs peuvent être omises en toute sécurité parce que le client va savoir que la progéniture d'une ressource ne pourrait pas être déplacée quand le client reçoit une erreur pour le parent. De plus, les réponses 201 (Créée) et 204 (Pas de contenu) NE DEVRAIENT PAS être retournées comme valeurs dans les réponses 207 (Multi-états) sur un MOVE. Ces réponses peuvent être omises en toute sécurité parce que elles sont les codes de succès par défaut.

9.9.3 MOVE et l'en-tête Overwrite

Si une ressource existe à la destination et si l'en-tête Overwrite est "T", alors avant d'effectuer le déplacement, le serveur DOIT effectuer un DELETE avec "Depth: infini" sur la ressource de destination. Si l'en-tête Overwrite est réglé à "F", l'opération va alors échouer.

9.9.4 Codes d'état

En plus des codes d'état généraux possibles, les codes d'état suivants ont une applicabilité spécifique pour MOVE :

201 (Créée) - la ressource de source a bien été déplacée, et une nouvelle transposition d'URL a été créée à la destination.

204 (Pas de contenu) - la ressource de source a bien été déplacée à un URL qui était déjà transposé.

207 (Multi-états) - plusieurs ressources devaient être affectées par le MOVE, mais des erreurs sur certaines d'entre elles ont

empêché l'opération d'avoir lieu. Des messages d'erreur spécifiques, ainsi que les URL de source et de destination les plus appropriés, apparaissent dans le corps de la réponse multi-états. Par exemple, si une ressource de source était verrouillée et ne pourrait pas être déplacée, alors l'URL de la ressource de source apparaîtra avec l'état 423 (Verrouillé).

403 (Interdit) - parmi les nombreuses raisons possibles pour interdire une opération MOVE, ce code d'état est recommandé quand la ressource de source et de destination est la même.

409 (Conflit) - une ressource ne peut pas être créée à la destination tant qu'une ou plusieurs collections intermédiaires n'ont pas été créées. Le serveur NE DOIT PAS créer automatiquement dcs collections intermédiaires. Ou, le serveur n'a pas été capable de préserver le comportement des propriétés vives et déplace quand même la ressource à la destination (voir la postcondition 'preserved-live-properties').

412 (Échec de précondition) - un en-tête de condition a échoué. Spécifiquement pour MOVE, cela pourrait signifier que l'en-tête Overwrite est "F" et que l'URL de destination est déjà transposé en une ressource.

423 (Verrouillé) - la ressource de source ou de destination, la ressource parente de source ou de destination, ou une certaine ressource dans la collection de source ou de destination, était verrouillée. Cette réponse DEVRAIT contenir l'élément de précondition 'lock-token-submitted'.

502 (Mauvaise passerelle) - cela peut arriver quand la destination est sur un autre serveur et que le serveur de destination refuse d'accepter la ressource. Cela pourrait aussi se produire quand la destination est sur une autre sous section du même espace de noms de serveur.

9.9.5 Exemple - MOVE pour une non collection

Cet exemple montre la ressource http://www.exemple.com/~fielding/index.html qui est déplacée à http://www.exemple.com/users/f/fielding/index.html. Le contenu de la ressource de destination aurait été écrasé si l'URL de destination avait déjà été transposé en une ressource. Dans ce cas, comme il n'y avait rien à la ressource de destination, le code de réponse est 201 (Créée).

>>Demande

```
MOVE /~fielding/index.html HTTP/1.1
Host: www.exemple.com
Destination: http://www.exemple/users/f/fielding/index.html
```

>>Réponse

```
HTTP/1.1 201 Créée
Location: http://www.exemple.com/users/f/fielding/index.html
```

9.9.6 Exemple - MOVE d'une collection

>>Demande

```
MOVE /conteneur/ HTTP/1.1
Host: www.exemple.com
Destination: http://www.exemple.com/autreconteneur/
Overwrite: F
If: (<urn:uuid:fe184f2e-6eec-41d0-c765-01adc56e6bb4>)
  (<urn:uuid:e454f3f3-acdc-452a-56c7-00a5c91e4b77>)
```

>>Réponse

```
HTTP/1.1 207 Multi-états
Content-Type: application/xml; charset="utf-8"
Content-Length: xxxx

<?xml version="1.0" encoding="utf-8" ?>
<d:multistatus xmlns:d='DAV:'>
 <d:response>
  <d:href>http://www.exemple.com/autreconteneur/C2/</d:href>
  <d:status>HTTP/1.1 423 Locked</d:status>
```

```
   <d:error><d:lock-token-submitted/></d:error>
  </d:response>
</d:multistatus>
```

Dans cet exemple, le client a soumis un certain nombre de jetons de verrou avec la demande. Un jeton de verrou va devoir être soumis pour chaque ressource, de source et de destination, partout dans la portée de la méthode, c'est-à-dire verrouillé. Dans ce cas, le jeton de verrou approprié n'a pas été soumis pour la destination http://www.exemple.com/autreconteneur/C2/. Cela signifie que la ressource /conteneur/C2/ ne pourrait pas être déplacée. Parce qu'il y a une erreur dans le déplacement de /conteneur/C2/, aucun des membres de /conteneur/C2 n'a été déplacé. Cependant, aucune erreur n'était mentionnée pour ces membres du fait de la règle de minimisation d'erreur. L'authentification d'agent d'utilisateur a eu lieu précédemment via un mécanisme qui sort du domaine du protocole HTTP dans une couche de transport sous-jacente.

9.10 Méthode LOCK

Les paragraphes qui suivent décrivent la méthode LOCK, qui est utilisée pour enlever un verrou de tout type d'accès et pour rafraîchir un verrou existant. Ces paragraphes sur la méthode LOCK décrivent seulement la sémantique qui est spécifique de la méthode LOCK et est indépendante du type d'accès du verrou demandé.

Toute ressource qui prend en charge la méthode LOCK DOIT, au minimum, prendre en charge la demande XML et les formats de réponse qui y sont définis.

Cette méthode n'est ni idempotente ni sûre (voir au paragraphe 9.1 de la [RFC2616]). Les réponses à cette méthode NE DOIVENT PAS être mises en antémémoire.

9.10.1 Création d'un verrou sur une ressource existanre

Une demande LOCK sur une ressource existante va créer un verrou sur la ressource identifiée par l'URI de demande, pourvu que la ressource ne soit pas déjà verrouillée avec un verrou en conflit. L'URI de demande devient la racine du verrou. Les demandes de méthode LOCK pour créer un nouveau verrou DOIVENT avoir un corps de demande XML. Le serveur DOIT préserver les informations fournies par le client dans l'élément 'owner' dans la demande LOCK. La demande LOCK PEUT avoir un en-tête Timeout.

Quand un nouveau verrou est créé, la réponse LOCK :
- o DOIT contenir un corps avec la valeur de la propriété DAV:lockdiscovery dans un élément XML prop. Celui-ci DOIT contenir les informations complètes sur le verrou qui vient d'être accordé, tandis que les informations sur les autres verrous (partagés) sont FACULTATIVES ;
- o DOIT inclure l'en-tête de réponse Lock-Token avec le jeton associé au nouveau verrou.

9.10.2 Rafraîchissement des verrous

Un verrou est rafraîchi par l'envoi d'une demande LOCK à l'URL d'une ressource dans la portée du verrou. Cette demande NE DOIT PAS avoir de corps et elle DOIT spécifier quel verrou rafraîchir en utilisant l'en-tête 'If' avec un seul jeton de verrou (un seul verrou peut être rafraîchi à la fois). La demande PEUT contenir un en-tête Timeout, qu'un serveur PEUT accepter de changer à la durée restante sur le verrou à la nouvelle valeur. Un serveur DOIT ignorer l'en-tête Depth sur un refraîchissement de LOCK.

Si la ressource a d'autres verrous (partagés) ces verrous ne sont pas affectés par un rafraîchissement de verrou. De plus, ces verrous n'empêchent pas le verrou désigné d'être rafraîchi.

L'en-tête Lock-Token n'est pas retourné dans la réponse pour une demande réussie de rafraîchissement de LOCK, mais le corps de réponse LOCK DOIT contenir la nouvelle valeur pour la propriété DAV:lockdiscovery.

9.10.3 En-tête Depth et verrouillage

L'en-tête Depth peut être utilisé avec la méthode LOCK. Les valeurs autres que 0 ou infini NE DOIVENT PAS être utilisées avec l'en-tête Depth sur une méthode LOCK. Toutes les ressources qui prennent en charge la méthode LOCK DOIVENT prendre en charge l'en-tête Depth.

Un en-tête Depth de valeur 0 signifie de juste verrouiller la ressource spécifiée par l'URI de demande.

Si l'en-tête Depth est réglé à infini, alors la ressource spécifiée dans l'URI de demande ainsi que tous ses membres, jusqu'au bout de la hiérarchie, sont à verrouiller. Un résultat réussi DOIT retourner un seul jeton de verrou. De même, si un

UNLOCK est exécuté avec succès sur ce jeton, toutes les ressources associées sont déverrouillées. Donc, un succès partiel n'est pas une option pour LOCK ou UNLOCK. Soit la hiérarchie entière est verrouillée, soit aucune ressource n'est verrouillée.

Si le verrou ne peut pas être accordé à toutes les ressources, le serveur DOIT retourner une réponse Multi-états avec un élément 'response' pour au moins une ressource qui a empêché le verrou d'être accordé, ainsi qu'un code d'état approprié pour cet échec (par exemple, 403 (Interdit) ou 423 (Verrouillé)). De plus, si la ressource qui cause l'échec n'était pas la ressource demandée, le serveur DEVRAIT alors aussi inclure un élément 'response' pour l'URI de demande, avec un élément 'status' contenant 424 (Échec de dépendance).

Si aucun en-tête Depth n'est soumis sur une demande LOCK, la demande DOIT alors agir comme si un "Depth:infini" avait été soumis.

9.10.4 Verrouillage d'URL non transposés

Une méthode LOCK réussie DOIT résulter en la création d'une ressource vide qui est verrouillée (et qui n'est pas une collection) quand une ressource n'existait pas précédemment à cet URL. Plus tard, le verrou peut partir mais la ressource vide reste. Les ressources vides DOIVENT alors apparaître dans les responses PROPFIND incluant cet URL dans la portée de réponse. Un serveur DOIT répondre par un succès à une demande GET sur une ressource vide, soit en utilisant une réponse 204 (Pas de contenu) soit en utilisant 200 OK avec un en-tête Content-Length indiquant une longueur de zéro.

9.10.5 Tableau de compatibilité de verrou

Le tableau ci-dessous décrit le comportement qui survient quand une demande de verrou est faîte sur une ressource.

État actuel	**Verrou partagé OK**	**Verrou exclusif OK**
Aucun	VRAI	VRAI
Verrou partagé	VRAI	FAUX
Verrou exclusif	FAUX	Faux*

Légende : Vrai = le verrou peut être accordé. Faux = le verrou NE DOIT PAS être accordé. *=Il est illégal pour un principal de demander deux fois le même verrou.

L'état de verrou courant d'une ressource est donné dans la colonne de gauche, et les demandes de verrou sont mentionnées dans la première rangée. L'intersection d'une rangée et d'une colonne donne le résultat d'une demande de verrou. Par exemple, si un verrou partagé est détenu sur une ressource, et si un verrou exclusif est demandé, l'entrée du tableau est "faux", ce qui indique que le verrou ne doit pas être accordé.

9.10.6 Réponses à LOCK

En plus des codes d'état généraux possibles, les codes d'état suivants ont une applicabilité spécifique pour LOCK:

200 (OK) - la demande LOCK a réussi et la valeur de la propriété DAV: lockdiscovery est incluse dans le corps de réponse.

201 (Créée) - la demande LOCK était à un URL non transposé, la demande a réussi et a résulté en la création d'une nouvelle ressource, et la valeur de la propriété DAV: lockdiscovery est incluse dans le corps de réponse

409 (Conflit) - une ressource ne peut pas être créée à la destination tant qu'une ou plusieurs collections intermédiaires n'ont pas été créées. Le serveur NE DOIT PAS créer automatiquement ces collections intermédiaires.

423 (Verrouillé), éventuellement avec le code de précondition 'no-conflicting-lock' - il y a déjà un verrou sur la ressource, qui est non compatible avec le verrou demandé (voir le tableau de compatibilité de verrou ci-dessus).

412 (Échec de précondition), avec le code de précondition 'lock-token-matches-request-uri' - la demande LOCK a été faite avec un en-tête If, qui indique que le client souhaite rafraîchir le verrou. Cependant, l'URI de demande ne rentre pas dans la portée du verrou identifié par le jeton. Le verrou peut avoir une portée qui n'inclut pas l'URI de demande, ou le verrou pourrait avoir disparu, ou le jeton peut être invalide.

9.10.7 Exemple - simple demande Lock

>>Demande

LOCK /workspace/webdav/proposal.doc HTTP/1.1

```
Host: exemple.com
Timeout: Infinite, Second-4100000000
Content-Type: application/xml; charset="utf-8"
Content-Length: xxxx
Authorization: Digest username="ejw",
  realm="ejw@exemple.com", nonce="...",
  uri="/workspace/webdav/proposal.doc",
  response="...", opaque="..."

<?xml version="1.0" encoding="utf-8" ?>
<D:lockinfo xmlns:D='DAV:'>
  <D:lockscope><D:exclusive/></D:lockscope>
  <D:locktype><D:write/></D:locktype>
  <D:owner>
    <D:href>http://exemple.org/~ejw/contact.html</D:href>
  </D:owner>
</D:lockinfo>
```

>>Réponse

```
HTTP/1.1 200 OK
Lock-Token: <urn:uuid:e71d4fae-5dec-22d6-fea5-00a0c91e6be4>
Content-Type: application/xml; charset="utf-8"
Content-Length: xxxx

<?xml version="1.0" encoding="utf-8" ?>
<D:prop xmlns:D="DAV:">
  <D:lockdiscovery>
    <D:activelock>
      <D:locktype><D:write/></D:locktype>
      <D:lockscope><D:exclusive/></D:lockscope>
      <D:depth>infinity</D:depth>
      <D:owner>
        <D:href>http://exemple.org/~ejw/contact.html</D:href>
      </D:owner>
      <D:timeout>Second-604800</D:timeout>
      <D:locktoken>
        <D:href
        >urn:uuid:e71d4fae-5dec-22d6-fea5-00a0c91e6be4</D:href>
      </D:locktoken>
      <D:lockroot>
        <D:href
        >http://exemple.com/workspace/webdav/proposal.doc</D:href>
      </D:lockroot>
    </D:activelock>
  </D:lockdiscovery>
</D:prop>
```

Cet exemple montre la création réussie d'un verrou en écriture exclusif sur une ressource http://exemple.com/workspace/webdav/proposal.doc. La ressource http://exemple.org/~ejw/contact.html contient des informations de contact pour le créateur du verrou. Le serveur a une politique fondée sur l'activité en place sur cette ressource, qui cause la suppression automatique du verrou après une semaine (604 800 secondes). Noter que les champs nonce, response, et opaque n'ont pas été calculés dans l'en-tête de demande Authorization.

9.10.8 Exemple – Rafraîchissement d'un verrou en écriture

>>Demande

```
LOCK /workspace/webdav/proposal.doc HTTP/1.1
Host: exemple.com
Timeout: Infinite, Second-4100000000
If: (<urn:uuid:e71d4fae-5dec-22d6-fea5-00a0c91e6be4>)
Authorization: Digest username="ejw",
```

```
  realm="ejw@exemple.com", nonce="...",
  uri="/workspace/webdav/proposal.doc",
  response="...", opaque="..."

>>Réponse

HTTP/1.1 200 OK
Content-Type: application/xml; charset="utf-8"
Content-Length: xxxx

<?xml version="1.0" encoding="utf-8" ?>
<D:prop xmlns:D="DAV:">
  <D:lockdiscovery>
    <D:activelock>
      <D:locktype><D:write/></D:locktype>
      <D:lockscope><D:exclusive/></D:lockscope>
      <D:depth>infinity</D:depth>
      <D:owner>
        <D:href>http://exemple.org/~ejw/contact.html</D:href>
      </D:owner>
      <D:timeout>Second-604800</D:timeout>
      <D:locktoken>
        <D:href
        >urn:uuid:e71d4fae-5dec-22d6-fea5-00a0c91e6be4</D:href>
      </D:locktoken>
      <D:lockroot>
        <D:href
        >http://exemple.com/workspace/webdav/proposal.doc</D:href>
      </D:lockroot>
    </D:activelock>
  </D:lockdiscovery>
</D:prop>
```

Cette demande va rafraîchir le verrou, tentant de réinitialiser le temporisateur à la nouvelle valeur spécifiée dans l'en-tête timeout. Noter que le client a demandé une temporisation infinie mais le serveur choisit d'ignorer la demande. Dans cet exemple, les champs nonce, response, et opaque n'ont pas été calculés dans l'en-tête de demande Authorization.

9.10.9 Exemple – Demande de verrou multi ressources

```
>>Demande

LOCK /webdav/ HTTP/1.1
Host: exemple.com
Timeout: Infinite, Second-4100000000
Depth: infinity
Content-Type: application/xml; charset="utf-8"
Content-Length: xxxx
Authorization: Digest username="ejw",
  realm="ejw@exemple.com", nonce="...",
  uri="/workspace/webdav/proposal.doc",
  response="...", opaque="..."

<?xml version="1.0" encoding="utf-8" ?>
<D:lockinfo xmlns:D="DAV:">
  <D:locktype><D:write/></D:locktype>
  <D:lockscope><D:exclusive/></D:lockscope>
  <D:owner>
    <D:href>http://exemple.org/~ejw/contact.html</D:href>
  </D:owner>
</D:lockinfo>

>>Réponse
```

```
HTTP/1.1 207 Multi-Status
Content-Type: application/xml; charset="utf-8"
Content-Length: xxxx

<?xml version="1.0" encoding="utf-8" ?>
<D:multistatus xmlns:D="DAV:">
  <D:response>
    <D:href>http://exemple.com/webdav/secret</D:href>
    <D:status>HTTP/1.1 403 Interdit</D:status>
  </D:response>
  <D:response>
    <D:href>http://exemple.com/webdav/</D:href>
    <D:status>HTTP/1.1 424 Échec de dépendance</D:status>
  </D:response>
</D:multistatus>
```

Cet exemple montre une demande pour un verrou en écriture exclusif sur une collection et tous ses enfants. Dans cette demande, le client a spécifié qu'il désire un verrou de longueur infinie, si disponible, autrement une temporisation de 4,1 milliards de secondes, si disponible. Le corps d'entité de demande contient les informations de contact pour le principal qui enlève le verrou -- dans ce cas, un URL de page de la Toile.

L'erreur est une réponse 403 (Interdit) sur la ressource http://exemple.com/webdav/secret. Parce que cette ressource ne pourrait pas être verrouillée, aucune des ressources n'a été verrouillée. Noter aussi qu'un élément 'response' pour l'URI de demande lui-même a été inclus comme exigé.

Dans cet exemple, les champs nonce, response, et opaque n'ont pas été calculés dans l'en-tête de demande Authorization.

9.11 Méthode UNLOCK

La méthode UNLOCK supprime le verrou identifié par le jeton de verrou dans l'en-tête de demande Lock-Token. L'URI de demande DOIT identifier une ressource dans la portée du verrou.

Noter que l'utilisation de l'en-tête Lock-Token pour fournir le jeton de verrou n'est pas cohérente avec les autres méthode de changement d'état, qui exigent toutes un en-tête If avec le jeton de verrou. Donc, l'en-tête If n'est pas nécessaire pour fournir le jeton de verrou. Naturellement, quand l'en-tête If est présent, il a sa signification normale d'en-tête conditionnel.

Pour une réponse de succès à cette méthode, le serveur DOIT supprimer entièrement le verrou.

Si toutes les ressources qui ont été verrouillées sous le jeton de verrou soumis ne peuvent pas être déverrouillées, la demande UNLOCK DOIT alors échouer.

Une réponse de succès à une méthode UNLOCK ne signifie pas que la ressource est nécessairement déverrouillée. Elle signifie que le verrou spécifique correspondant au jeton spécifié n'existe plus.

Toute ressource conforme à DAV qui prend en charge la méthode LOCK DOIT prendre en charge la méthode UNLOCK.

Cette méthode est idempotente, mais non sûre (voir au paragraphe 9.1 de la [RFC2616]). Les réponses à cette méthode NE DOIVENT PAS être mises en antémémoire.

9.11.1 Codes d’état

En plus des codes d’état généraux possibles, les codes d’état suivants ont une applicabilité spécifique pour UNLOCK :

204 (Pas de contenu) - réponse normale de succès (plutôt que 200 OK, car 200 OK impliquerait un corps de réponse, et une réponse de succès de UNLOCK ne contient normalement pas de corps).

400 (Mauvaise demande) - aucun jeton de verrou n'a été fourni.

403 (Interdit) -le principal actuellement authentifié n'a pas la permission de supprimer le verrou.

409 (Conflit) avec la précondition 'lock-token-matches-request-uri' - la ressource n'était pas verrouillée, ou la demande a été faite à un URI de demande tqui n'était pas dans la portée du verrou.

9.11.2 Exemple - UNLOCK

>>Demande

```
UNLOCK /workspace/webdav/info.doc HTTP/1.1
Host: exemple.com
Lock-Token: <urn:uuid:a515cfa4-5da4-22e1-f5b5-00a0451e6bf7>
Authorization: Digest username="ejw"
  realm="ejw@exemple.com", nonce="...",
  uri="/workspace/webdav/proposal.doc",
  response="...", opaque="..."
```

>>Réponse

```
HTTP/1.1 204 Pas de contenu
```

Dans cet exemple, le verrou identifié par le jeton de verrou "urn:uuid:a515cfa4-5da4-22e1-f5b5-00a0451e6bf7" est bien supprimé de la ressource http://exemple.com/workspace/webdav/info.doc. Si ce verrou inclut plus que juste une ressource, le verrou est supprimé de toutes les ressources incluses dans le verrou.

Dans cet exemple, les champs nonce, response, et opaque n'ont pas été calculés dans l'en-tête de demande Authorization.

10. En-têtes HTTP pour la distribution des noms d’auteurs

Tous les en-têtes DAV suivent les mêmes règles de base de formatage que les en-têtes HTTP. Cela inclut des règles comme la continuation de ligne et comment combiner (ou séparer) plusieurs instances du même en-tête en utilisant des virgules.

WebDAV ajoute deux nouveaux en-têtes conditionnels à l'ensemble défini dans HTTP : les en-têtes If et Overwrite.

10.1 En-tête DAV

```
DAV = "DAV" ":" #( classe de conformité )
   classe de conformité = ( "1" | "2" | "3" | extend )
   extend = URL-codé | jeton                    ; jeton est défini au paragraphe 2.2 de la RFC 2616.
   URL-codé = "<" URI-absolu ">"
                ; Pas d'espace linéaire (LWS) permise dans URL-codé URI-absolu définis au paragraphe 4.3 de la RFC 3986.
```

L'apparition de cet en-tête général dans la réponse indique que la ressource prend en charge le schéma et protocole DAV comme spécifié. Toutes les ressources conformes à DAV DOIVENT retourner l'en-tête DAV avec la classe de conformité "1" sur toutes les réponses OPTIONS. Dans les cas où WebDAV est seulement pris en charge dans une partie de l'espace de noms du serveur, une demande OPTIONS pour des ressources non WebDAV (incluant "/") NE DEVRAIT PAS annoncer la prise en charge de WebDAV.

La valeur est une liste séparée par des virgules de tous les identifiants de classe de conformité que la ressource prend en charge. Les identifiants de classe peuvent être des URL codés ou des jetons (comme défini par la [RFC2616]). Les identifiants peuvent apparaître dans n'importe quel ordre. Les identifiants qui sont normalisés par le processus des RFC de l'IETF sont des jetons, mais les autres identifiants DEVRAIENT être des URL-codés pour favoriser l'unicité.

Une ressource doit afficher la conformité de classe 1 si elle affiche la conformité de classe 2 ou 3. En général, la prise en charge d'une classe de conformité n'entraîne pas la prise en charge d'une autre, et en particulier, la prise en charge de la conformité de classe 3 n'exige pas la conformité de classe 2. Voir à la Section 18 plus de détails sur les classes de conformité définies dans la présente spécification.

Noter que de nombreux serveurs WebDAV n'annoncent pas la prise en charge de WebDAV en réponse à "OPTIONS *".

Comme en-tête de demande, cet en-tête permet au client d'annoncer la conformité à des caractéristiques désignées quand le serveur a besoin de cette information. Les clients NE DEVRAIENT PAS envoyer cet en-tête sauf si une spécification sur la voie de la normalisation l'exige. Toute extension qui utilise cela comme un en-tête de demande aura besoin d'examiner avec attention les implications d'une mise en antémémoire.

10.2 En-tête Depth

Depth = "Depth" ":" ("0" | "1" | "infini")

L'en-tête de demande Depth est utilisé avec les méthodes exécutées sur des ressources qui pourraient avoir des membres internes pour indiquer si la méthode est à appliquer seulement à la ressource ("Depth: 0"), à la ressource et seulement ses membres internes ("Depth: 1"), ou à la ressource et tous ses membres ("Depth: infini").

L'en-tête Depth est seulement pris en charge si la définition d'une méthode fournit explicitement cette prise en charge.

Les règles suivantes sont la valeur du comportement par défaut pour toute méthode qui prend en charge l'en-tête Depth. Une méthode peut outrepasser ces valeurs par défaut en définissant un comportement different.

Les méthodes qui prennent en charge l'en-tête Depth peuvent choisir de ne pas prendre en charge toutes les valeurs d'en-tête et peuvent définir, cas par cas, le comportement de la méthode si un en-tête Depth n'est pas présent. Par exemple, la méthode MOVE prend seulement en charge "Depth: infini", et si un en-tête Depth n'est pas présent, elle va agir comme si un en-tête "Depth: infini" avait été appliqué.

Les clients NE DOIVENT PAS compter que les méthodes s'exécutent sur les membres de leur hiérarchie dans un ordre particulier ou que l'exécution soit atomique sauf si la méthode particulière fournit explicitement de telles garanties.

Lors de l'exécution, une méthode avec un en-tête Depth va effectuer autant de la tâche qui lui est allouée que possible et ensuite retourner une réponse spécifiant ce qu'elle a été capable d'accomplir et ce qu'elle a échoué à faire.

Ainsi, par exemple, une tentative de COPY d'une hiérarchie peut résulter en ce que certains des membres soient copiés et d'autres non.

Par défaut, l'en-tête Depth n'interagit pas avec d'autres en-têtes. C'est-à-dire, chaque en-tête sur une demande avec un en-tête Depth DOIT être appliqué seulement à l'URI de demande si il s'applique à toute ressource, sauf si un comportement Depth spécifique est défini pour cet en-tête.

Si une ressource de source ou destination dans la portée de l'en-tête Depth est verrouillée de façon à empêcher la réussite de l'exécution de la méthode, alors le jeton de verrou pour cette ressource DOIT être soumis avec la demande dans l'en-tête If de la demande.

L'en-tête Depth spécifie seulement le comportement de la méthode à l'égard des membres internes. Si une ressource n'a pas de membres internes, alors l'en-tête Depth DOIT être ignoré.

10.3 En-tête Destination

L'en-tête de demande Destination spécifie l'URI qui identifie une ressource de destination pour des méthodes comme COPY et MOVE, qui prennent deux URI comme paramètres.

Destination = "Destination" ":" Simple-ref

Si la valeur de Destination est un URI absolu (paragraphe 4.3 de la [RFC3986]) elle peut désigner un serveur different (ou un accès ou schéma différent). Si le serveur de source ne peut pas tenter la copie sur le serveur distant, il DOIT faire échouer la demande. Noter que la copie et le déplacement de ressources sur des serveurs distants ne sont pas pleinement définis dans la présente spécification (par exemple, des conditions d'erreur spécifiques).

Si la valeur de Destination est trop longue ou autrement inacceptable, le serveur DEVRAIT retourner 400 (Mauvaise demande) idéalement avec des informations utiles dans un corps d'erreur.

10.4 En-tête If

L'en-tête de demande If est destiné à avoir une fonction similaire à l'en-tête If-Match défini au paragraphe 14.24 de la [RFC2616]. Cependant, l'en-tête If traite tous les jetons d'état ainsi que les ETag. Une exemple typique d'un jeton d'état est un jeton de verrou, et les jetons de verrou sont les seuls jetons d'état définis dans la présente spécification.

10.4.1 Objet

L'en-tête If a deux objets distincts :

- Le premier objet est de rendre une demande conditionnelle en fournissant une série de listes d'états avec des conditions qui correspondent aux jetons et ETag pour une ressource spécifique. Si cet en-tête est évalué et si tous les états de la liste échouent, la demande DOIT alors échouer avec un état 412 (Échec de précondition). Par ailleurs, la demande ne peut réussir que si une des listes d'état décrites réussit. Les critères de succès des listes d'états et des fonctions correspondantes sont définis aux paragraphes 10.4.3 et 10.4.4.
- De plus, le simple fait qu'un jeton d'état apparaisse dans un en-tête If signifie qu'il a été "soumis" avec la demande. En général, c'est utilisé pour indiquer que le client a connaissance de ce jeton d'état. La sémantique de la soumission d'un jeton d'état dépend de son type (pour les jetons de verrou, voir la Section 6).

Noter que ces deux objets doivent être traités séparément : un jeton d'état compte comme étant soumis indépendamment de si le serveur a en fait évalué la liste d'états dans laquelle il apparaît, et aussi indépendamment de si la condition qu'il exprime se trouve ou non être vraie.

10.4.2 Syntaxe

```
If = "If" ":" ( 1*No-tag-list | 1*Tagged-list )
No-tag-list = List
Tagged-list = Resource-Tag 1*List

List = "(" 1*Condition ")"
Condition = ["Not"] (State-token | "[" entity-tag "]")
; entity-tag : voir au paragraphe 3.11 de la [RFC2616]
; Aucune espace n'est permise entre "[", entity-tag et "]"

State-token = Coded-URL

Resource-Tag = "<" Simple-ref ">"
; Simple-ref : voir au paragraphe 8.3
; Aucune espace n'est permise dans Resource-Tag
```

La syntaxe distingue entre les listes non étiquetées ("No-tag-list") et les listes étiquetées ("Tagged-list"). Les listes non étiquetées s'appliquent à la ressource identifiée par l'URI de demande, tandis que les listes étiquetées s'appliquent à la ressource identifiée par l'étiquette de ressource précédente.

Une étiquette de ressource s'applique à toutes les listes suivantes, jusqu'à la prochaine étiquette de ressource.

Noter que les deux types de listes ne peuvent pas être mêlés dans un en-tête If. Ce n'est pas une restriction fonctionnelle parce que la syntaxe de No-tag-list est juste une notation abrégée pour une production Tagged-list avec une Resource-Tag se référant à l'URI de demande.

Chaque liste consiste en une ou plusieurs conditions. Chaque condition est définie dans les termes d'une étiquette d'entité ou de jeton d'état, potentiellement niée par le préfixe "Not".

Noter que la syntaxe de l'en-tête If ne permet pas plusieurs instances d'en-têtes If dans une seule demande. Cependant, la syntaxe d'en-tête HTTP permet d'étendre des valeurs d'un seul en-tête sur plusieurs lignes, en insérant une coupure de ligne suivie par des espaces (voir le paragraphe 4.2 de la [RFC2616]).

10.4.3 Évaluation de liste

Une condition qui consiste en une seule étiquette d'entité ou jeton d'état s'évalue à vrai si la ressource correspond à l'état décrit (où les fonctions individuelles de correspondance sont définies au paragraphe 10.4.4). Un préfixe de "Not" inverse le résultat de l'évaluation (donc, le "Not" s'applique seulement à l'étiquette d'entité ou jeton d'état suivant).

Chaque production List décrit une série de conditions. La liste entière s'évalue à vrai si et seulement si chaque condition s'évalue à vrai (c'est-à-dire, la liste représente une conjonction logique de conditions).

Chaque production No-tag-list et Tagged-list peut contenir une ou plusieurs listes. Elle s'évalue à vrai si et seulement si toutes les listes contenues s'évaluent à vrai (c'est-à-dire, si il y a plus d'une liste, cette séquence de listes représente une disjonction logique des listes).

Finalement, l'en-tête If entier s'évalue à vrai si et seulement si au moins une des productions No-tag-list ou Tagged-list

s'évalue à vrai. Si l'en-tête s'évalue à faux, le serveur DOIT rejeter la demande avec un état 412 (Échec de précondition). Autrement, l'exécution de la demande peut se poursuivre comme si l'en-tête n'était pas présent.

10.4.4 Confrontation des jetons d'état et des ETag

Quand on effectue le traitement de l'en-tête If, la définition d'un jeton d'état ou d'étiquette d'entité correspondante est comme suit :

Identification d'une ressource : la ressource est identifiée par l'URI avec le jeton, dans la production d'une liste étiquetée, ou par l'URI de demande dans une production de liste non étiquetée.

Correspondance d'étiquette d'entité : lorsque l'étiquette d'entité correspond à une étiquette d'entité associée à la ressource identifiée. Les serveurs DOIVENT utiliser la fonction de comparaison faible ou forte définie au paragraphe 13.3.3 de la [RFC2616].

Correspondance de jeton d'état : lorsque il y a une correspondance exacte entre le jeton d'état dans l'en-tête If et tout jeton d'état sur la ressource identifiée. Un jeton d'état verrouillé est considéré correspondre si la ressource est dans la portée du verrou.

Traitement des URL non transposés : pour les ETag et les jetons d'état, les traiter comme si l'URL identifiait une ressource qui existe mais n'a pas l'état spécifié.

10.4.5 En-tête If et mandataires sans capacité DAV

Les mandataires sans capacité DAV ne vont pas honorer l'en-tête If, car ils ne vont pas comprendre l'en-tête If, et HTTP exige que les en-têtes non compris soient ignorés. Quand il communique avec des mandataires HTTP/1.1, le client DOIT utiliser l'en-tête de demande "Cache-Control: no-cache" afin d'empêcher que le mandataire essaye à tort de servir la demande à partir de son antémémoire. Quand on traite avec des mandataires HTTP/1.0, l'en-tête de demande "Pragma: no-cache" DOIT être utilisé pour la même raison.

Parce qu'en général les clients peuvent n'être pas capables de détecter fiablement les intermédiaires sans capacité DAV, il est conseillé de toujours empêcher la mise en antémémoire en utilisant les directives de demande mentionnées ci-dessus.

10.4.6 Exemple – Pas de production d'étiquette

```
If: (<urn:uuid:181d4fae-7d8c-11d0-a765-00a0c91e6bf2>
     ["Je suis une ETag"])
     (["Je suis une autre ETag"])
```

L'en-tête précédent exigerait que la ressource identifiée dans l'URI de demande soit verrouillée avec le jeton de verrou spécifié et soit dans l'état identifié par la ETag "Je suis une ETag" ou dans l'état identifié par la seconde ETag "Je suis une autre ETag".

Pour dire les choses plus clairement, on peut voir le premier en-tête If comme exprimant la condition ci-dessous :

```
(
    est-verrouillé-avec(urn:uuid:181d4fae-7d8c-11d0-a765-00a0c91e6bf2) ET
    correspond-à-l'etag("Je suis une ETag")
  )
  OU
  (
    correspond-à-l'etag("Je suis une autre ETag")
  )
```

10.4.7 Exemple – Utilisation de "Not" sans production d'étiquette

```
If: (Not <urn:uuid:181d4fae-7d8c-11d0-a765-00a0c91e6bf2>
         <urn:uuid:58f202ac-22cf-11d1-b12d-002035b29092>)
```

Cet en-tête If exige que la ressource ne soit pas verrouillée avec un verrou ayant le jeton de verrou urn:uuid:181d4fae-7d8c-11d0-a765-00a0c91e6bf2 et doit être verrouillée par un verrou avec le jeton de verrou urn:uuid:58f202ac-22cf-11d1-b12d-002035b29092.

10.4.8 Exemple – Faire qu'une condition s'évalue toujours à Vrai

Il peut y avoir des cas où un client souhaite soumettre des jetons d'état, mais ne veut pas que la demande échoue juste parce que le jeton d'état n'est plus en cours. Une façon simple de faire cela est d'inclure une condition qui est connue pour toujours s'évaluer à vrai, comme dans :

```
If: (<urn:uuid:181d4fae-7d8c-11d0-a765-00a0c91e6bf2>)
    (Not <DAV:no-lock>)
```

"DAV:no-lock" est connu pour ne jamais représenter un jeton de verrou en cours. Les jetons de verrou sont alloués par le serveur, suivant les exigences d'unicité décrites au paragraphe 6.5, donc ils ne peuvent pas utiliser le schéma "DAV:". Donc, en appliquant "Not" à un jeton d'état qui est connu pour n'être pas en cours, la condition s'évalue toujours à vrai. Par conséquent, l'en-tête If entier va toujours s'évaluer à vrai, et le jeton de verrou urn:uuid:181d4fae-7d8c-11d0-a765-00a0c91e6bf2 va être soumis dans tous les cas.

10.4.9 Exemple - En-tête If de liste étiquetée dans COPY

>>Demande

```
COPY /ressource1 HTTP/1.1
Host: www.exemple.com
Destination: /ressource2
If: </ressource1>
  (<urn:uuid:181d4fae-7d8c-11d0-a765-00a0c91e6bf2>
  [W/"Une ETag faible"]) (["ETag forte"])
```

Dans cet exemple, http://www.exemple.com/ressource1 est copié à http://www.exemple.com/ressource2. Quand la méthode est d'abord appliquée à http://www.exemple.com/ressource1, ressource1 doit être dans l'état spécifié par "(<urn:uuid:181d4fae-7d8c-11d0-a765-00a0c91e6bf2> [W/"Une ETag faible"]) (["ETag forte"])". C'est-à-dire, soit elle doit être verrouillée avec un jeton de verrou de "urn:uuid:181d4fae-7d8c-11d0-a765-00a0c91e6bf2" et avoir une étiquette d'entité faible W/"Une ETag faible", soit elle doit avoir une étiquette d'entité forte "ETag forte".

10.4.10 Exemple - Confrontation de jetons de verrou avec des verrous de collection

```
DELETE /specs/rfc2518.txt HTTP/1.1
Host: www.exemple.com
If: <http://www.exemple.com/specs/>
  (<urn:uuid:181d4fae-7d8c-11d0-a765-00a0c91e6bf2>)
```

Pour cet exemple, le jeton de verrou doit être comparé à la ressource identifiée, qui est la collection 'specs' identifiée par l'URL dans la production de liste étiquetée. Si la collection 'specs' n'est pas verrouillée par un verrou avec le jeton de verrou specifié, la demande DOIT échouer. Autrement, cette demande pourrait réussir, parce que l'en-tête If s'évalue à vrai, et parce que le jeton de verrou pour le verrou qui affecte la ressource concernée a été soumis.

10.4.11 Exemple - Confrontation de ETag sur des URL non transposés

Considérons une collection "/specs" qui ne contient pas le membre "/specs/rfc2518.doc". Dans ce cas, l'en-tête If

```
If: </specs/rfc2518.doc> (["4217"])
```

va s'évaluer à faux (l'URI n'est pas transposé, donc la ressource identifiée par l'URI n'a pas une entité correspondant à la ETag "4217").

Par ailleurs, un en-tête If de If: </specs/rfc2518.doc> (Not ["4217"]) va par conséquent s'évaluer à vrai.

Noter que, comme défini au paragraphe 10.4.4, les mêmes considerations s'appliquent à la confrontation des jetons d'état.

10.5 En-tête Lock-Token

Lock-Token = "Lock-Token" ":" URL codé

L'en-tête de demande Lock-Token est utilisé avec la méthode UNLOCK pour identifier le verrou à supprimer. Le jeton de verrou dans l'en-tête de demande Lock-Token DOIT identifier un verrou qui contient comme membre la ressource

identifiée par l'URI de demande.

L'en-tête de demande Lock-Token est utilisé avec la méthode LOCK pour indiquer le jeton de verrou créé par suite de la réussite d'une demande LOCK de créer un nouveau verrou.

10.6 En-tête Overwrite

Overwrite = "Overwrite" ":" ("T" | "F")

L'en-tête de demande Overwrite spécifie si le serveur devrait écraser une ressource transposée en l'URL de destination durant une opération COPY ou MOVE. Une valeur de "F" déclare que le serveur ne doit pas effectuer l'opération COPY ou MOVE si l'URL de destination ne se transpose pas en une ressource. Si l'en-tête Overwrite n'est pas inclus dans une demande COPY ou MOVE, alors la ressource DOIT traiter la demande comme si elle avait un en-tête Overwrite de valeur "T". Alors que l'en-tête Overwrite apparaît pour dupliquer la fonction d'utilisation d'un en-tête "If-Match: *" (voir la [RFC2616]) If-Match s'applique seulement à l'URI de demande, et non à la destination d'un COPY ou MOVE.

Si un COPY ou MOVE n'est pas effectué du fait de la valeur de l'en-tête Overwrite, la méthode DOIT échouer avec un code d'état 412 (Échec de précondition). Le serveur DOIT faire des vérifications d'autorisation avant de vérifier cela ou tout en-tête conditionnel.

Toutes les ressources conformes à DAV DOIVENT prendre en charge l'en-tête Overwrite.

10.7 En-tête de demande Timeout

TimeOut = "Timeout" ":" 1#TimeType
TimeType = ("Second-" DAVTimeOutVal | "Infini") ; pas d'espaces permises dans TimeType.
DAVTimeOutVal = 1*DIGIT

Les clients PEUVENT inclure des en-têtes de demande Timeout dans leurs demandes LOCK. Cependant, le serveur n'est pas obligé d'honorer ou même considérer ces demandes. Les clients NE DOIVENT PAS soumettre un en-tête de demande Timeout avec une méthode autre que LOCK.

Le type de temps "Second" spécifie le nombre de secondes qui va s'écouler entre l'allocation du verrou au serveur, et la suppression automatique du verrou. La valeur de temporisation pour le TimeType "Second" NE DOIT PAS être supérieure à 2^32-1.

Voir au paragraphe 6.6 la description du comportement de temporisation de verrou.

11. Extensions de codes d'état à HTTP/1.1

Les codes d'état suivants sont ajoutés à ceux définis dans HTTP/1.1 [RFC2616].

11.1 207 Multi-états

Le code d'état 207 (Multi-états) donne l'état pour plusieurs opérations indépendantes (voir plus d'informations à la Section 13).

11.2 422 Entité non traitable

Le code d'état 422 (Entité non traitable) signifie que le serveur comprend le type de contenu de l'entité de demande (donc un code d'état 415 (Type de support non pris en charge) n'est pas approprié) et la syntaxe de l'entité de demande est correcte (donc un code d'état 400 (Mauvaise demande) est inapproprié) mais qu'il a été incapable de traiter les instructions contenues. Par exemple, cette condition d'erreur peut survenir si un corps de demande XML contient des instructions XML bien formées (c'est-à-dire, syntaxiquement correctes) mais sémantiquement erronées.

11.3 423 Verrouillé

Le code d'état 423 (Verrouillé) signifie que la ressource de source ou destination d'une méthode est verrouillée. Cette réponse DEVRAIT contenir un code de précondition ou postcondition approprié, comme 'lock-token-submitted' ou 'no-

conflicting-lock'.

11.4 424 Échec de dépendance

Le code d'état 424 (Échec de dépendance) signifie que la méthode ne pourrait pas être effectuée sur la ressource parce que l'action demandée dépend d'une autre action et que cette action a échoué. Par exemple, si une commande échoue dans une méthode PROPPATCH, alors, au minimum, le reste de la commande va aussi échouer avec un 424 (Échec de dépendance).

11.5 507 Mémoire insuffisante

Le code d'état 507 (Mémoire insuffisante) signifie que la méthode ne pourrait pas être effectuée sur la ressource parce que le serveur est incapable de mémoriser la représentation nécessaire pour réussir à achever la demande. Cette condition est considérée comme temporaire. Si la demande qui a reçu ce code d'état était le résultat de l'action d'un utilisateur, la demande NE DOIT PAS être répétée jusqu'à ce qu'ells soit demandée par une action d'utilisateur distincte.

12. Utilisation des codes d’état HTTP

Ces codes HTTP ne sont pas redéfinis, mais leur utilisation est un peu étendue par les méthodes et exigences WebDAV. En général, de nombreux codes d’état HTTP peuvent être utilisés en réponse à toute demande, pas juste dans les cas décrits dans le présent document. Noter aussi que les serveurs WebDAV sont connus pour utiliser des réponses de redirection de niveau 300 (et des essais d'interopérabilité précoces ont trouvé des clients qui ne sont pas prêts à traiter ces réponses). Une réponse de niveau 300 NE DOIT PAS être utilisée quand le serveur a créé une nouvelle ressource en réponse à la demande.

12.1 412 Échec de précondition

Toute demande peut contenir un en-tête conditionnel défini dans HTTP (If-Match, If-Modified-Since, etc.) ou les en-têtes conditionnels "If" ou "Overwrite" définis dans la présente spécification. Si le serveur évalue un en-tête conditionnel, et si cette condition ne tient pas, ce code d'erreur DOIT alors être retourné. Par ailleurs, si le client n'a pas inclus d'en-tête conditionnel dans la demande, alors le serveur NE DOIT PAS utiliser ce code d'état.

12.2 414 URI de demande trop long

Ce code d'état est utilisé dans HTTP 1.1 seulement pour les URI de demande, pas pour les URI dans d'autres localisations.

13. Réponses multi états

Une réponse multi-états porte des informations sur plusieurs ressources dans des situations où plusieurs codes d’état pourraient être appropriés. Le code de réponse multi-états par défaut est une entité HTTP text/xml ou application/xml avec un élément racine 'multistatus'. Les éléments suivants contiennent des codes d'état des séries 200, 300, 400, et 500 générés durant l'invocation de la méthode. Les codes d’état de la série 100 NE DEVRAIENT PAS être enregistrés dans un élément XML 'response'.

Bien que '207' soit utilisé comme code d'état de réponse global, le receveur a besoin de consulter le contenu du corps de la réponse multi-états pour avoir plus d'informations sur le succès ou l'échec de l'exécution de la méthode. La réponse PEUT être utilisée dans des situations de succès, de succès partiel et aussi d'échec.

L'élément racine 'multistatus' contient zéro, un ou plusieurs éléments 'response' dans un ordre quelconque, chacun avec des informations sur une ressource individuelle. Chaque élément 'response' DOIT avoir un élément 'href' pour identifier la ressource.

Une réponse Multi-états utilise un des deux formats distincts suivants pour représenter l'état :

1. Un élément 'status' comme fils de l'élément 'response' indique l'état de l'exécution du message pour la ressource identifiée comme un tout (par exemple, voir au paragraphe 9.6.2). Certaines définitions de méthode fournissent des informations sur des codes d’état spécifiques que les clients devraient être prêts à voir dans une réponse. Cependant, les clients DOIVENT être capables de traiter les autres codes d’état, en utilisant les règles générales définies à la Section 10 de la [RFC2616].

2. Pour PROPFIND et PROPPATCH, le format a été étendu en utilisant l'élément 'propstat' au lieu de 'status', fournissant des informations sur les propriétés individuelles d'une ressource. Ce format est spécifique de PROPFIND et PROPPATCH, et est décrit en détails aux paragraphes 9.1 et 9.2.

13.1 En-tête de réponse

HTTP définit l'en-tête Location pour indiquer un URL préféré pour la ressource qui était visée dans l'URI de demande (par exemple, en réponse à des demandes PUT réussies ou dans des réponses de redirection). Cependant, l'utilisation de cet en-tête crée une ambiguïté quand il y a des URL dans le corps de la réponse, comme avec Multi-états. Donc, l'utilisation de l'en-tête Location avec la réponse Multi-états est intentionellement indéfini.

13.2 Traitement de ressources filles redirigées

Les réponses redirigées (300-303, 305, et 307) définies dans HTTP 1.1 prennent normalement un en-tête Location pour indiquer le nouvel URI pour la seule ressource redirigée à partir de l'URI de demande. Les réponses Multi-états contiennent de nombreuses adresses de ressources, mais la définition originale dans la [RFC2518] ne laissait aucune place pour que le serveur fournisse le nouvel URI pour les ressources redirigées. La présente spécification définit un élément 'location' pour cette information (voir au paragraphe 14.9). Les serveurs DOIVENT utiliser ce nouvel élément avec les réponses de redirection dans Multi-états.

Les clients qui rencontrent des ressources redirigées dans Multi-états NE DOIVENT PAS s'appuyer sur la présence de l'élément 'location' avec un nouvel URI. Si l'élément n'est pas présent, le client PEUT réitérer la demande à la ressource individuelle redirigée, parce que la réponse à cette demande peut être redirigée avec un en-tête Location contenant le nouvel URI.

13.3 Codes d’état internes

Les paragraphes 9.1.2, 9.2.1, 9.6.1, 9.8.3, et 9.9.2 définissent divers codes d’état utilisés dans les réponses Multi-états. La présente spécification ne définit pas la signification des autres codes d’état qui pourraient apparaître dans ces réponses.

14. Définitions d’élément XML

Dans cette section, la ligne finale de chaque paragraphe donne la déclaration de type d'élément (DTD) en utilisant le format défini dans [XML]. Le champ "Valeur", lorsque présent, spécifie des restrictions supplémentaires sur le contenu admis pour l'élément XML en utilisant le BNF (c’est-à-dire, pour restreindre les valeurs d'un élément PCDATA). Noter que tous les éléments définis ici peuvent être étendus en accord avec les règles définies à la Section 17. Tous les éléments définis ici sont dans l'espace de noms "DAV:".

14.1 Élément XML activelock

Nom : activelock
Objet : décrit un verrou sur une ressource.

<!ELEMENT activelock (lockscope, locktype, depth, owner?, timeout?, locktoken?, lockroot)>

14.2 Élément XML allprop

Nom : allprop
Objet : spécifie que tous les noms et valeurs des propriétés mortes et vives définies par le présent document existant sur la ressource sont à retourner.

!ELEMENT allprop EMPTY >

14.3 Élément XML collection

Nom : collection

Objet : identifie la ressource associée comme une collection. La propriété DAV:ressourcetype d'une ressource de collection DOIT contenir cet élément. Il est normalement vide mais des extensions peuvent ajouter des sous éléments.

<!ELEMENT collection EMPTY >

14.4 Élément XML depth

Nom : depth
Objet : utilisé pour représenter les valeurs de profondeur dans le contenu XML (par exemple, dans les informations de verrou).
Valeur : "0" | "1" | "infini"

<!ELEMENT depth (#PCDATA) >

14.5 Élément XML error

Nom : error
Objet : les réponses d'erreur, en particulier 403 Interdit et 409 Conflit, ont parfois besoin de plus d'informations pour indiquer ce qui ne va pas. Dans ce cas, les serveurs PEUVENT retourner un corps de réponse XML avec un élément de document de 'error', contenant les éléments fils qui identifient des codes de condition particuliers.

Description : contient au moins un élément XML, et NE DOIT PAS contenir de texte ou de contenu mixte. Tout élément qui est un fils de l'élément 'error' est considéré être un code de précondition ou de postcondition. Les éléments non reconnus DOIVENT être ignorés.

<!ELEMENT error ANY >

14.6 Élément XML exclusive

Nom : exclusive
Objet : spécifie un verrou exclusif.

<!ELEMENT exclusive EMPTY >

14.7 Élément XML href

Nom : href
Objet : DOIT contenir un URI ou une référence relative.
Description : il peut y avoit des limites à la valeur de 'href' selon le contexte de son utilisation. Se référer au texte de spécification où 'href' est utilisé pour voir quelles limitations s'appliquent dans chaque cas.
Valeur : Simple-ref

<!ELEMENT href (#PCDATA)>

14.8 Élément XML include

Nom : include
Objet : tout élément fils représente le nom d'une propriété à inclure dans la réponse PROPFIND. Tous les éléments dans un élément XML 'include' DOIT définir les propriétés relatives à la ressource, bien que de possible noms de propriété ne soient en aucune façn limités à ces noms de propriété définis dans le présent document ou d'autres normes. Cet élément NE DOIT PAS contenir de contenu de texte ou mixte.

<!ELEMENT include ANY >

14.9 Élément XML location

Nom : location
Objet : HTTP définit l'en-tête "Location" (voir le paragraphe 14.30 de la [RFC2616]) à utiliser avec certains codes d'état (comme 201 et les codes de la série 300). Quand ces codes sont utilisés dans un élément 'multistatus', l'élément

'location' peut être utilisé pour fournir la valeur de l'en-tête Location qui l'accompagne.
Description : contient un seul élément href avec la même valeur que celle qui serait utilisée dans un en-tête Location.

<!ELEMENT location (href)>

14.10 Élément XML lockentry

Nom : lockentry
Objet : Définit les types de verrous qui peuvent être utilisés avec la ressource.

<!ELEMENT lockentry (lockscope, locktype) >

14.11 Élément XML lockinfo

Nom : lockinfo
Objet : l'élément XML 'lockinfo' est utilisé avec une méthode LOCK pour spécifier le type de verrou que le client souhaite créer.

<!ELEMENT lockinfo (lockscope, locktype, owner?) >

14.12 Élément XML lockroot

Nom : lockroot
Objet : contient l'URL racine du verrou, qui est l'URL par lequel la ressource a été adressée dans la demande LOCK.
Description : l'élément href contient la racine du verrou. Le serveur DEVRAIT l'inclure dans toutes lesvaleurs de propriété DAV:lockdiscovery et dans la réponse aux demandes LOCK.

<!ELEMENT lockroot (href) >

14.13 Élément XML lockscope

Nom : lockscope
Objet : spécifie si un verrou est exclusif ou partagé.

<!ELEMENT lockscope (exclusive | shared) >

14.14 Élément XML locktoken

Nom : locktoken
Objet : jeton de verrou associé à un verrou.
Description : la href contient un seul URI de jeton de verrou, qui se réfère au verrou.

<!ELEMENT locktoken (href) >

14.15 Élément XML locktype

Nom : locktype
Objet : spécifie le type d'accès d'un verrou. Pour l'instant, la présente spécification définit seulement un type de verrou, le verrou en écriture.

<!ELEMENT locktype (write) >

14.16 Élément XML multistatus

Nom : multistatus
Objet : contient plusieurs messages de réponse.
Description : l'élément 'responsedescription' est utilisé au niveau supérieur pour fournir un message général décrivant la nature de la réponse. Si cette valeur est disponible, une application peut l'utiliser au lieu de présenter les descriptions de réponses individuelles contenues dans les réponses.

<!ELEMENT multistatus (response*, responsedescription?) >

14.17 Élément XML owner

Nom : owner
Objet : contient les informations fournies par un client sur le créateur d'un verrou.
Description : permet à un client de fournir des informations suffisantes pour soit contacter directement un principal (comme un numéro de téléphone ou un URI de messagerie électrtonique) soit découvrir le principal (comme l'URL d'une page d'accueil) qui a créé un verrou. La valeur fournie DOIT être traitée comme une propriété morte en termes de préservation d'élément d'information XML. Le serveur NE DOIT PAS altérer la valeur sauf si la valeur de owner fournie par le client est vide. Pour un certain niveau d'interopérabilité entre les différentes mises en œuvre de client, si les clients ont des informations de contact en forme d'URI pour le créateur de verrou qui conviennent pour l'affichage à l'utilisateur, les clients DEVRAIENT mettre ces URI dans les éléments fils 'href' de l'élément 'owner'.
Extensibilité : PEUT être étendu avec des éléments fils, du contenu mixte, du contenu de texte ou des attributs.

<!ELEMENT owner ANY >

14.18 Élément XML prop

Nom : prop
Objet : contient des propriétés relatives à une ressource.
Description : conteneur générique pour les propriétés définies sur des ressources. Tous les éléments dans un élément XML 'prop' DOIVENT définir les propriétés relatives à la ressource, bien que les noms des propriétés possibles ne soient en aucune façon limités aux noms de propriétés définis dans le présent document ou d'autres normes. Cet élément NE DOIT PAS contenir de texte ou de contenu mixte.

<!ELEMENT prop ANY >

14.19 Élément XML propertyupdate

Nom : propertyupdate
Objet : contient une demande d'altérer les propriétés sur une ressource.
Description : cet élément XML est un conteneur pour les informations nécessaires pour modifier les propriétés sur la ressource.

<!ELEMENT propertyupdate (remove | set)+ >

14.20 Élément XML propfind

Nom : propfind
Objet : spécifie les propriétés à retourner d'une méthode PROPFIND. Quatre éléments spéciaux sont spécifiés pour être utilisés avec 'propfind' : 'prop', 'allprop', 'include', et 'propname'. Si 'prop' est utilisé dans 'propfind', il NE DOIT PAS contenir de valeurs de propriété.

<!ELEMENT propfind (propname | (allprop, include?) | prop) >

14.21 Élément XML propname

Nom : propname
Objet : spécifie qu'une seule liste de noms de propriétés de la ressource est à retourner.

<!ELEMENT propname EMPTY >

14.22 Élément XML propstat

Nom : propstat
Objet : grouper un élément prop et un élément status qui sont associés à un élément 'href' particulier.
Description : l'élément XML propstat DOIT contenir un élément XML prop et un élément XML status. Le contenu de

l'élément XML prop DOIT seulement faire la liste des noms des propriétés auxquelles le résultat de l'élément status s'applique. L'élément facultatif precondition/postcondition et le texte 'responsedescription' s'applique aussi aux propriétés désignées dans 'prop'.

<!ELEMENT propstat (prop, status, error?, responsedescription?) >

14.23 Élément XML remove

Nom : remove
Objet : fait la liste des propriétés à supprimer d'une ressource.
Description : "remove" donne pour instruction que les propriétés spécifiées dans "prop" devraient être supprimées. Spécifier la suppression d'une propriété qui n'existe pas n'est pas une erreur. Tous les éléments XML dans un élément XML 'prop' à l'intérieur d'un élément XML 'remove' DOIVENT être vides, car seuls les noms des propriétés à supprimer sont exigés.

<!ELEMENT remove (prop) >

14.24 Élément XML response

Nom : response
Objet : contient une seule réponse décrivant l'effet d'une méthode sur une ressource et/ou ses propriétés.
Description : l'élément 'href' contient un URL HTTP pointant sur une ressource WebDAV quand il est utilisé dans le conteneur 'response'. Une valeur 'href' particulière NE DOIT PAS apparaître plus d'une fois comme enfant d'un élément XML 'response' sous un élément XML 'multistatus'. Cette exigence est nécessaire afin de garder les coûts de traitement d'une réponse à un temps linéaire. Essentiellement, cela empêche d'avoir à chercher afin de grouper toutes les réponses par 'href'. Il n'y a cependant aucune exigence sur l'ordre fondée sur les valeurs de 'href'. L'élément facultatif précondition/postcondition et le texte 'responsedescription' peuvent fournir des informations supplémentaires sur cette ressource relativement à la demande ou son résultat.

<!ELEMENT response (href, ((href*, status)|(propstat+)), error?, responsedescription? , location?) >

14.25 Élément XML responsedescription

Nom : responsedescription
Objet : contient des information sur une réponse d'état dans un Multi-états.
Description : fournit des informations convenables pour être présentées à un utilisateur.

<!ELEMENT responsedescription (#PCDATA) >

14.26 Élément XML set

Nom : set
Objet : fait la liste des valeurs de propriété à établir pour une ressource.
Description : l'élément 'set' DOIT contenir seulement un élément 'prop'. Les éléments contenus par l'élément 'prop' dans l'élément 'set' DOIVENT spécifier le nom et la valeur des propriétés qui sont établies sur la ressource identifiée par l'URI de demande. Si une propriété existe déjà, alors sa valeur est remplacée. Les informations d'étiquetage de langage qui apparaissent dans la portée de l'élément 'prop' (dans l'attribut "xml:lang", si il est présent) DOIVENT être mémorisées de façon persistante avec la propriété, et DOIVENT être restituables ultérieurement en utilisant PROPFIND.

<!ELEMENT set (prop) >

14.27 Élément XML shared

Nom : shared
Objet : spécifie un verrou partagé.

<!ELEMENT shared EMPTY >

14.28 Élément XML status

Nom : status
Objet : contient une seule ligne d'état HTTP.
Valeur : status-line (définie au paragraphe 6.1 de la [RFC2616])

<!ELEMENT status (#PCDATA) >

14.29 Élément XML timeout

Nom : timeout
Objet : nombre de secondes restantes avant l'expiration d'un verrou.
Valeur : TimeType (définie au paragraphe 10.7)

<!ELEMENT timeout (#PCDATA) >

14.30 Élément XML write

Nom : write
Objet : spécifie un verrou en écriture.

<!ELEMENT write EMPTY >

15. Propriétés DAV

Pour les propriétés DAV, le nom de la propriété est aussi le même que le nom de l'élément XML qui contient sa valeur. Dans la présente Section, la dernière ligne de chaque paragraphe donne la déclaration de type d'élément en utilisant le format défini dans [XML]. Le champ "Valeur", lorsque il est présent, spécifie plus de restrictions sur les contenus admissibles de l'élément XML en utilisant le BNF (c'est-à-dire, pour restreindre les valeurs d'un élément PCDATA).

Une propriété protégée est celle qui ne peut pas être changée avec une demande PROPPATCH. Il peut y avoir d'autres demandes qui vont résulter en un changement d'une propriété protégée (comme quand une demande LOCK affecte la valeur de DAV:lockdiscovery). Noter qu'une propriété donnée pourrait être protégée sur un type de ressource, mais pas sur un autre type de ressource.

Une propriété calculée est celle qui a une valeur définie en termes de calcul (sur la base du contenu et d'autres propriétés de cette ressource, ou même d'une autre ressource). Une propriété calculée est toujours une propriété protégée.

Les comportements COPY et MOVE se réfèrent aux opérations locales COPY et MOVE.

Pour les propriétés définies sur la base des en-têtes de réponse HTTP GET (DAV:get*) la valeur d'en-tête pourrait inclure des espaces blanches, comme défini au paragraphe 4.2 de la [RFC2616]. Les mises en œuvre de serveur DEVRAIENT supprimer les espaces de ces valeurs avant de les utiliser comme valeurs de propriété WebDAV.

15.1 Propriété creationdate

Nom : creationdate
Objet : enregistre la date et l'heure de création de la ressource.
Valeur : date-time (définie dans la [RFC3339], voir l'ABNF au paragraphe 5.6.)
Protégée : PEUT être protégée. Certains serveurs permettent que DAV:creationdate soit changé pour refléter l'heure de création du document si elle est plus significative pour l'utilisateur (plutôt que l'heure de son téléchargement). Donc, les clients NE DEVRAIENT PAS utiliser cette propriété dans la logique de synchronisation (utiliser plutôt DAV:getetag).
Comportement COPY/MOVE : cette valeur de propriété DEVRAIT être conservée durant une opération MOVE, mais est normalement réinitialisée quand une ressource est créée avec COPY. Elle ne devrait pas être établie dans un COPY.
Description : la propriété DAV:creationdate DEVRAIT être définie sur toutes les ressources conformes à DAV. Si elle est présente, elle contient un horodatage du moment de la création de la ressource. Les serveurs qui sont incapables d'enregistrer de façon persistante la date de création DEVRAIENT plutôt la laisser indéfinie (c'est-à-dire rapporter "Non trouvé").

<!ELEMENT creationdate (#PCDATA) >

15.2 Propriété displayname

Nom : displayname
Objet : fournit un nom pour la ressource qui convient pour la présentation à un utilisateur.
Valeur : tout texte.
Protégée : NE DEVRAIT PAS être protégée. Noter que les serveurs qui mettent en œuvre la [RFC2518] pourraient en avoir fait une propriété protégée car c'est une nouvelle exigence.
Comportement COPY/MOVE : cette valeur de propriété DEVRAIT être préservée dans les opérations COPY et MOVE.
Description : contient une description de la ressource qui convient pour la présentation à un utilisateur. Cette propriété est définie sur la ressource, et donc DEVRAIT avoir la même valeur indépendamment de l'URI de demande utilisé pour la restituer (donc, calculer cette propriété sur la base de l'URI de demande est déconseillé). Bien que les clients génériques puissent afficher la valeur de propriété aux utilisateurs finaux, les concepteurs d'URI de client doivent comprendre que la méthode pour identifier les ressources est toujours l'URL. Les changements à DAV:displayname ne produisent pas de déplacements ou de copies au serveur, mais changent simplement un élément de métadonnées sur la ressource individuelle. Deux ressources peuvent avoir la même valeur de DAV:displayname même au sein de la même collection.

<!ELEMENT displayname (#PCDATA) >

15.3 Propriété getcontentlanguage

Nom : getcontentlanguage
Objet : contient la valeur de l'en-tête Content-Language (d'après le paragraphe 14.12 dela [RFC2616]) comme elle serait retournée par un GET sans les en-têtes Accept.
Valeur : language-tag (language-tag est défini au paragraphe 3.10 de la [RFC2616])
Protégé : NE DEVRAIT PAS être protégée, afin que les clients puissent réinitialiser le langage. Noter que les serveurs qui mettent en œuvre la [RFC2518] pourraient l'avoir rendue une propriété protégée car c'est une nouvelle exigence.
Comportement COPY/MOVE : cette valeur de propriété DEVRAIT être préservée dans les opérations COPY et MOVE.
Description : la propriété DAV:getcontentlanguage DOIT être définie sur toute ressource conforme à DAV qui retourne l'en-tête Content-Language sur un GET.

<!ELEMENT getcontentlanguage (#PCDATA) >

15.4 Propriété getcontentlength

Nom : getcontentlength
Objet : contient l'en-tête Content-Length retourné par un GET sans en-tête Accept.
Valeur : voir au paragraphe 14.13 de la [RFC2616].
Protégée : cette propriété est calculée, donc protégée.
Description : la propriété DAV:getcontentlength DOIT être définie sur toute ressource conforme à DAV qui retoue l'en-tête Content-Length en réponse à un GET.
Comportement COPY/MOVE : cette valeur de propriété dépend de la taille de la ressource de destination, non de la valeur de la propriété sur la ressource source

<!ELEMENT getcontentlength (#PCDATA) >

15.5 Propriété getcontenttype

Nom : getcontenttype
Objet : contient la valeur d'en-tête Content-Type (d'après le paragraphe 14.17 de la [RFC2616]) comme elle serait retournée par un GET sans en-tête Accept.
Valeur : media-type (défini au paragraphe 3.7 de la [RFC2616])
Protégée : potentiellement protégée si le serveur préfère allouer des types de contenu à lui (voir aussi la discussion du paragraphe 9.7.1).
Comportement COPY/MOVE : cette valeur de propriété DEVRAIT être préservée dans les opérations COPY et MOVE.
Description : cette propriété DOIT être définie sur toute ressource conforme à DAV qui retourne l'en-tête Content-Type en réponse à un GET.

<!ELEMENT getcontenttype (#PCDATA) >

15.6 Propriété getetag

Nom : getetag
Objet : contient la valeur de l'en-tête ETag (d'après le paragraphe 14.19 de la [RFC2616]) comme elle serait retournée par un GET sans en-tête Accept.
Valeur : entity-tag (définie au paragraphe 3.11 de la [RFC2616])
Protégée : DOIT être protégée parce que cette valeur est créée et contrôlée par le serveur.
Comportement COPY/MOVE : cette valeur de propriété dépend de l'état final de la ressource de destination, et non de la valeur de la propriété sur la ressource source. Noter aussi les considérations du paragraphe 8.8.
Description : la propriété getetag DOIT être définie sur toute ressource conforme à DAV qui retourne l'en-tête Etag. Voir au paragraphe 3.11 de la RFC 2616 une définition complète de la sémantique d'une ETag, et au paragraphe 8.6 une discussion des ETag dans WebDAV.

<!ELEMENT getetag (#PCDATA) >

15.7 Propriété getlastmodified

Nom : getlastmodified
Objet : contient la valeur d'en-tête Last-Modified (d'après le paragraphe 14.29 de la [RFC2616]) comme elle serait retournée par une méthode GET sans en-tête Accept.
Valeur : rfc1123-date (définie au paragraphe 3.3.1 de la [RFC2616])
Protégée : DEVRAIT être protégée parce que certains clients peuvent s'appuyer sur la valeur pour un comportement approprié de mise en antémémoire, ou sur la valeur de l'en-tête Last-Modified auquel cette propriété est liée.
Comportement COPY/MOVE : cette valeur de propriété dépend de la date de dernière modification de la ressource de destination, non de la valeur de la propriété sur la ressource source. Noter que certaines mises en œuvre de serveur utilisent la valeur modifiée de date de système de fichier pour la valeur DAV:getlastmodified, et cela peut être préservé dans un MOVE même quand la valeur HTTP Last-Modified DEVRAIT changer. Note que comme la [RFC2616] exige des clients qu'ils utilisent des ETag lorsque elles sont fournies, un serveur qui met en œuvre les ETag peut compter que les clients utilisent un bien meilleur mécanisme que les dates de modification pour la synchronisation hors ligne ou le contrôle d'antémémoire. Noter aussi les considérations du paragraphe 8.8.
Description : la date last-modified sur une ressource DEVRAIT seulement refléter les changements dans le corps (les réponses GET) de la ressource. Un changement dans une propriété seulement NE DEVRAIT PAS causer le changement de la date de dernière modification, parce que les clients PEUVENT s'appuyer sur la date de dernière modification pour savoir quand écraser le corps existant. La propriété DAV: getlastmodified DOIT être définie sur toute ressource conforme à DAV qui retourne l'en-tête Last-Modified en réponse à un GET.

<!ELEMENT getlastmodified (#PCDATA) >

15.8 Propriété lockdiscovery

Nom : lockdiscovery
Objet : décrit les verrous actifs sur une ressource
Protégée : DOIT être protégée. Les clients changent la liste des verrous avec LOCK et UNLOCK, mais pas avec PROPPATCH.
Comportement COPY/MOVE : la valeur de cette propriété dépend de l'état de la destination du verrou, et non des verrous sur la ressource source. On rappelle que les verrous ne sont pas déplacés dans une opération MOVE.
Description : retourne une liste de qui a un verrou, quel type de verrou il a, le type de temporisation et le temps restant sur la temporisation, et le jeton de verrou associé. Les informations de propriétaire PEUVENT être omises si elles sont considérées comme sensibles. Si il n'y a pas de verrou, mais si le serveur prend en charge les verrous, la propriété va être présente mais contiendra zéro élément 'activelock'. Si il y a un ou plusiurs verrous, un élément 'activelock' apparaît pour chaque verrou sur la ressource. Cette propriété N'EST PAS verrouillable par rapport aux verrous en écriture (Section 7).

<!ELEMENT lockdiscovery (activelock)* >

15.8.1 Exemple – Restitution de DAV:lockdiscovery

>>Demande

PROPFIND /conteneur/ HTTP/1.1
Host: www.exemple.com
Content-Length: xxxx

```
Content-Type: application/xml; charset="utf-8"

<?xml version="1.0" encoding="utf-8" ?>
<D:propfind xmlns:D='DAV:'><D:prop><D:lockdiscovery/></D:prop>
</D:propfind>

>>Réponse

HTTP/1.1 207 Multi-Status
Content-Type: application/xml; charset="utf-8"
Content-Length: xxxx

<?xml version="1.0" encoding="utf-8" ?>
<D:multistatus xmlns:D='DAV:'>
  <D:response>
    <D:href>http://www.exemple.com/conteneur/</D:href>
    <D:propstat>
      <D:prop>
        <D:lockdiscovery>
          <D:activelock>
            <D:locktype><D:write/></D:locktype>
            <D:lockscope><D:exclusive/></D:lockscope>
            <D:depth>0</D:depth>
            <D:owner>Jane Smith</D:owner>
            <D:timeout>Infinite</D:timeout>
            <D:locktoken>
              <D:href>urn:uuid:f81de2ad-7f3d-a1b2-4f3c-00a0c91a9d76</D:href>
            </D:locktoken>
            <D:lockroot>
              <D:href>http://www.exemple.com/conteneur/</D:href>
            </D:lockroot>
          </D:activelock>
        </D:lockdiscovery>
      </D:prop>
      <D:status>HTTP/1.1 200 OK</D:status>
    </D:propstat>
  </D:response>
</D:multistatus>
```

Cette ressource a un seul verrou en écriture exclusif sur elle, avec une temporisation infinie.

15.9 Propriété ressourcetype

Nom : ressourcetype

Objet : spécifie la nature de la ressource.

Protégée : DEVRAIT être protégée. Le type de ressource est généralement décidé par l'opération qui crée la ressource (MKCOL ou PUT) pas par PROPPATCH.

Comportement COPY/MOVE : généralement un COPY/MOVE d'une ressource résulte en le même type de ressource à la destination.

Description : DOIT être définie sur toutes les ressources conformes à DAV. Chaque élément fils identifie le type spécifique auquel la ressource appartient, comme 'collection', qui est le seul type de ressource défini par la présente spécification (voir au paragraphe 14.3). Si l'élément contient l'élément fils 'collection' plus des éléments supplémentaires non reconnus, il devrait généralement être traité comme une collection. Si l'élément ne contient pas d'éléments fils reconnus, il devrait être traité comme une ressource non de collection. La valeur par défaut est vide. Cet élément NE DOIT PAS contenir de texte ou contenu mixte. Tout élément fils personnalisé est considéré comme étant un identifiant d'un type de ressource.

```
Exemple : (exemple fictif pour montrer l'extensibilité)
<x:ressourcetype xmlns:x="DAV:">
  <x:collection/>
  <f:search-results xmlns:f="http://www.exemple.com/ns"/>
</x:ressourcetype>
```

15.10 Propriété supportedlock

Nom : supportedlock
Objet : fournir une liste des capacités de verrou prises en charge par la ressource.
Protégée : DOIT être protégée. Les serveurs, non les clients, déterminent quels mécanismes de verrou sont supportés.
Comportement COPY/MOVE : cette valeur de propriété dépend de la sorte de verrous supportés à la destination, non de la valeur de la propriété à la ressource source. Les serveurs qui tentent une opération COPY sur une destination ne devraient pas tenter d'établir cette propriété à la destination.
Description : retourne une liste des combinaisons des portées et types d'accès qui peuvent être spécifiées dans une demande de verrou sur la ressource. Noter que les contenus réels sont eux-mêmes contrôlés par des contrôles d'accès, de sorte qu'un serveur n'est pas obligé de fournir des informations que le client n'est pas autorisé à voir. Cette propriété N'EST PAS verrouillable par rapport aux verrous en écriture (Section 7).

<!ELEMENT supportedlock (lockentry)* >

15.10.1 Exemple – Restitution de DAV:supportedlock

>>Demande

```
PROPFIND /conteneur/ HTTP/1.1
Host: www.exemple.com
Content-Length: xxxx
Content-Type: application/xml; charset="utf-8"

<?xml version="1.0" encoding="utf-8" ?>
<D:propfind xmlns:D="DAV:"><D:prop><D:supportedlock/></D:prop>
</D:propfind>
```

>>Réponse

```
HTTP/1.1 207 Multi-Status
Content-Type: application/xml; charset="utf-8"
Content-Length: xxxx

<?xml version="1.0" encoding="utf-8" ?>
<D:multistatus xmlns:D="DAV:">
  <D:response>
    <D:href>http://www.exemple.com/conteneur/</D:href>
    <D:propstat>
      <D:prop>
        <D:supportedlock>
          <D:lockentry>
            <D:lockscope><D:exclusive/></D:lockscope><D:locktype><D:write/></D:locktype>
          </D:lockentry>
          <D:lockentry>
            <D:lockscope><D:shared/></D:lockscope><D:locktype><D:write/></D:locktype>
          </D:lockentry>
        </D:supportedlock>
      </D:prop>
      <D:status>HTTP/1.1 200 OK</D:status>
    </D:propstat>
  </D:response>
</D:multistatus>
```

16. Éléments XML Precondition/Postcondition

Comme indiqué au paragraphe 8.7, des informations supplémentaires sur les conditions d'erreur peuvent être incluses dans le corps de nombreuses réponses d'état. Cette Section formule les exigences sur l'utilisation du mécanisme de corps d'erreur et introduit un certain nombre de codes de précondition et postcondition.

Une "précondition" d'une méthode décrit l'état du serveur qui doit être vrai pour que cette méthode soit effectuée. Une "postcondition" d'une méthode décrit l'état du serveur qui doit être vrai après que cette méthode a été effectuée.

Chaque précondition et postcondition a un élément XML unique associé. Dans une réponse 207 Multi-états, l'élément XML DOIT apparaître dans un élément 'error' dans l'élément approprié 'propstat ou 'response' selon que la condition s'applique à une ou plusieurs propriétés ou à la ressource comme un tout. Dans toutes les autres réponses d'erreur où le corps 'error' de la présente spécification est utilisé, l'élément XML précondition/postcondition DOIT être retourné comme fils d'un élément 'error' de niveau supérieur dans le corps de la réponse, sauf négocié autrement par la demande, avec un état de réponse approprié. Les codes d'état de réponse les plus courants sont 403 (Interdit) si la demande ne devrait pas être répétée parce que elle va toujours échouer, et 409 (Conflit) si on s'attend à ce que l'utilisateur pourrait être capable de résoudre le conflit et resoumettre la demande. L'élément 'error' PEUT contenir des éléments fils avec des informations spécifiques de l'erreur et PEUT être étendue avec tous éléments fils spécialisés.

Ce mécanisme ne remplace pas l'utilisation d'un code d'état numérique correct comme défini ici ou dans HTTP, parce que le client doit toujours être capable de prendre une action raisonnable sur la seule base du code numérique. Cependant, il ne supprime pas le besoin de définir de nouveaux codes numériques. Les nouveaux codes lisibles par la machine utilisés à cette fin sont des éléments XML classés comme préconditions et postconditions, donc naturellement, tout groupe qui définit un nouveau code de condition peut utiliser son propre espace de noms. Comme toujours, l'espace de noms "DAV:" est réservé à l'usage des groupes de travail WebDAV mandatés par l'IETF.

Un serveur qui prend en charge la présente spécification DEVRAIT utiliser l'erreur XML chaque fois qu'une précondition ou postcondition définie dans le présent document est violée. Pour les conditions d'erreur non spécifiées dans le présent document, le serveur PEUT simplement choisir un état numérique approprié et laisser blanc le corps de réponse. Cependant, un serveur PEUT utiliser à la place un code de condition personnalisé et un texte de support autre, parce que même quand les clients ne reconnaissent pas automatiquement les codes de condition, ils peuvent être assez utiles dans les essais d'interopérabilité et le débogage.

Exemple - Réponse avec code de précondition

>>Réponse

```
HTTP/1.1 423 Locked
Content-Type: application/xml; charset="utf-8"
Content-Length: xxxx

<?xml version="1.0" encoding="utf-8" ?>
<D:error xmlns:D="DAV:">
  <D:lock-token-submitted>
    <D:href>/workspace/webdav/</D:href>
  </D:lock-token-submitted>
</D:error>
```

Dans cet exemple, un client ignorant d'un verrou de profondeur infinie sur la collection parente "/workspace/webdav/" a tenté de modifier le membre de collection "/workspace/webdav/proposal.doc".

Certaines autres préconditions et postconditions utiles ont été définies dans d'autres spécifications qui étendent WebDAV, comme la [RFC3744] (voir en particulier le paragraphe 7.1.1), la [RFC3253], et la [RFC3648].

Tous ces éléments sont dans l'espace de noms "DAV:". Sauf spécification contraire, le contenu de chaque élément XML de condition est défini comme étant vide.

Nom : lock-token-matches-request-uri
Utilisé avec : 409 Conflit
Objet : (précondition) -- une demande peut inclure un en-tête Lock-Token pour identifier un verrou pour la méthode UNLOCK. Cependant, si l'URI de demande ne rentre pas dans la portée du verrou identifié par le jeton, le serveur DEVRAIT utiliser cette erreur. Le verrou peut avoir une portée qui n'inclut pas l'URI de demande, ou le verrou pourrait avoir disparu, ou le jeton peut être invalide.

Nom : lock-token-submitted (precondition)
Utilisé avec : 423 Verrouillé
Objet : la demande pourrait ne pas réussir parce que un jeton de verrou aurait dû être soumis. Cet élément, si il est présent, DOIT contenir au moins un URL d'une ressource verrouillée qui a empêché la demande. Dans les cas de MOVE,

COPY, et DELETE où des verrous de collection sont impliqués, il peut être difficile au client de trouver quelle ressource verrouillée a fait échouer la demande -- mais le serveur est seulement chargé de retourner une telle ressource verrouillée. Le serveur PEUT retourner chaque ressource verrouillée qui a empêché la demande de réussir si il les connaît toutes.

<!ELEMENT lock-token-submitted (href+) >

Nom : no-conflicting-lock (precondition)
Utilisé avec : normalement 423 Verrouillée
Objet : une demande LOCK a échoué à cause de la présence d'un verrou en conflit déjà existant. Noter qu'un verrou peut être en conflit bien que la ressource à laquelle la demande était dirigée soit seulement indirectement verrouillée. Dans ce cas, le code de précondition peut être utilisé pour informer le client sur la ressource qui est la racine du verrou en conflit, évitant une recherche séparée de la propriété "lockdiscovery".

<!ELEMENT no-conflicting-lock (href)* >

Nom : no-external-entities
Utilisé avec : 403 Interdit
Objet : (précondition) -- si le serveur rejette la demande d'un client parce que le corps de la demande contient une entité externe, le serveur DEVRAIT utiliser cette erreur.

Nom : preserved-live-properties
Utilisé avec : 409 Conflit
Objet : (postcondition) -- le serveur a reçu une demande MOVE ou COPY par ailleurs valide, mais ne peut pas maintenir les propriétés vives avec le même comportement à la destination. Il se peut que le serveur prenne en charge seulement certaines propriétés vives dans certaines parties du répertoire, ou ait simplement une erreur interne.

Nom : propfind-finite-depth
Utilisé avec : 403 Interdit
Objet : (précondition) -- ce serveur ne permet pas les demandes PROPFIND de profondeur infinie sur les collections.

Nom : cannot-modify-protected-property
Utilisé avec : 403 Interdit
Objet : (précondition) -- le client a tenté d'établir une propriété protégée dans une PROPPATCH (comme DAV:getetag). Voir aussi au paragraphe 3. 12 de la [RFC3253].

17 Extensibilité XML dans DAV

L'extension d'espace de noms XML [XML-NAMES] est utilisée dans la présente spécification afin de permettre d'ajouter de nouveaux éléments XML sans crainte de collision avec d'autres noms d'éléments. Bien que les corps de demande et de réponse WebDAV puissent être étendus par des éléments XML arbitraires, qui peuvent être ignorés par le receveur du message, un élément XML dans l'espace de noms "DAV:" NE DEVRAIT PAS être utilisé dans le corps de la demande ou de la réponse sauf si cet élément XML est explicitement défini dans une RFC de l'IETF revue par un groupe de travail WebDAV.

Pour que WebDAV soit à la fois extensible et rétro-compatible, les clients et les serveurs doivent savoir comment se comporter quand des extensions de commandes inattendues ou non reconnues sont reçues. Pour le traitement XML, cela signifie que les clients et serveurs DOIVENT traiter les documents XML reçus comme si des éléments et attributs inattendus (et tous les enfants des éléments non reconnus) n'étaient pas là. Un élément ou attribut inattendu inclut celui qui peut être utilisé dans un autre contexte mais n'est pas attendu à cet endroit. Ignorer de tels éléments au moment du traitement peut bien sûr être cohérent avec l'enregistrement de toutes les informations ou leur présentation pour le débogage.

Cette restriction s'applique aussi au traitement, par les clients, des valeurs de propriété DAV lorsque des éléments XML inattendus DEVRAIENT être ignorés sauf si le sctéma de propriété en décide autrement.

Cette restriction ne s'applique pas à l'établissement de propriétés DAV mortes sur le serveur quand le serveur DOIT enregistrer tous les éléments XML.

De plus, cette restriction ne s'applique pas à l'utilisation de XML lorsque XML se trouve être le type de contenu du corps d'entité, par exemple, quand utilisé comme corps d'un PUT.

Le traitement des instructions en XML DEVRAIT être ignoré par les receveurs. Donc, les spécifications qui étendent

WebDAV NE DEVRAIENT PAS utiliser le traitement des instructions pour définir un comportement normatif.

Les fragments de DTD XML sont inclus pour tous les éléments XML définis dans la présente spécification. Cependant, le XML correct ne sera pas valide pour tout DTD à cause de l'usage de l'espace de noms et des règles d'extension. En particulier :

- o les éléments (de la présente spécification) sont dans l'espace de noms "DAV:",
- o l'ordre des éléments n'est pas significatif sauf mention contraire,
- o des attributs d'extension PEUVENT être ajoutés,
- o pour les définitions de type d'élément de "ANY", la définition de texte normatif pour cet élément définit ce qui peut être dans elle et ce que cela signifie,
- o pour les définitions de type d'élément de "#PCDATA", des éléments d'extension NE DOIVENT PAS être ajoutés,
- o pour les autres définitions de type d'élément, incluant "EMPTY", des éléments d'extension PEUVENT être ajoutés.

Noter que cela signifie que les éléments qui contiennent des éléments ne peuvent pas être étendus à contenir du texte, et vice versa.

Avec la validation de DTD assouplie par les règles ci-dessus, les contraintes décrites par les fragments de DTD sont normatives (voir par exemple l'Appendice A). Le receveur d'un message WebDAV avec un corps XML NE DOIT PAS valider le document XML selon un DTD incorporé ou déclaré dynamiquement.

Noter que cette Section décrit les règles d'extensibilité rétro-compatibles. Il pourrait aussi y avoir des cas où une extension est conçue comme n'étant pas rétro-compatible, par exemple, de définir une extension qui réutilise un élément XML défini dans le présent document mais omettant un des éléments fils requis par les DTD de la présente spécification.

18. Classes de conformité DAV

Une ressource conforme à DAV peut annoncer plusieurs classes de conformité. Un client peut découvrir les classes de conformité d'une ressource en exécutant OPTIONS sur la ressource et en examinant l'en-tête "DAV" retourné. Noter particulièrement que les ressources, plutôt que les serveurs, sont vues comme étant conformes. C'est parce que théoriquement certaines ressources sur un serveur pourraient prendre en charge différents ensembles de caractéristiques. Par exemple, un serveur pourrait avoir un sous répertoire où une caractéristique avancée comme la prise en compte des versions serait prise en charge, même si cette caractéristique n'est pas prise en charge sur tous les sous répertoires.

Comme le présent document décrit des extensions au protocole HTTP/1.1, au minimum toutes les ressources, clients, et mandataires conformes à DAV DOIVENT être conformes à la [RFC2616].

Une ressource qui est conforme à la classe 2 ou à la classe 3 doit aussi être conforme à la classe 1.

18.1 Classe 1

Une ressource conforme à la classe 1 DOIT satisfaire toutes les exigences marquées "DOIT" dans tous les paragraphes du présent document.

Les ressources conformes à la classe 1 DOIVENT retourner, au minimum, la valeur "1" dans l'en-tête DAV sur toutes les réponses à la méthode OPTIONS.

18.2 Classe 2

Une ressource conforme à la classe 2 DOIT satisfaire toutes les exigences de classe 1 et prendre en charge la méthode LOCK, la propriété DAV:supportedlock, la propriété DAV: lockdiscovery, l'en-tête de réponse Time-Out et l'en-tête de demande Lock-Token. Une ressource conforme à la classe 2 DEVRAIT aussi prendre en charge l'en-tête de demande Timeout et l'élément XML 'owner'.

Les ressources conformes à la classe 2 DOIVENT retourner, au minimum, les valeurs "1" et "2" dans l'en-tête DAV sur toutes les réponses à la méthode OPTIONS.

18.3 Classe 3

Une ressource peut explicitement annoncer sa prise en charge des révisions à la [RFC2518] faites dans le présent

document. La classe 1 DOIT aussi être prise en charge. La classe 2 PEUT être prise en charge. L'annonce de la prise en charge de la classe 3 en plus des classes 1 et 2 signifie que le serveur prend en charge toutes les exigences de la présente spécification. Annoncer la prise en charge de la classe 3 et de la classe 1, mais pas de la classe 2, signifie que le serveur prend en charge toutes les exigences de la présente spécification sauf éventuellement celles qui impliquent le verrouillage.

Exemple : DAV: 1, 3

19. Considérations d'internationalisation

Dans le domaine de l'internationalisation, la présente spécification se conforme à la politique des jeux de caractères de l'IETF [RFC2277]. Dans la présente spécification, les champs lisibles par l'homme se trouvent soit dans la valeur d'une propriété, soit dans un message d'erreur retourné dans un corps d'entité de réponse. Dans les deux cas, le contenu lisible par l'homme est codé en utilisant XML, qui a des dispositions explicites pour l'étiquetage des jeux de caractères et du codage, et exige que les processeurs XML lisent les éléments XML codés, au minimum, en utilisant les codages UTF-8 [RFC3629] et UTF-16 [RFC2781] du plan multilingue de la norme ISO 10646. Les exemples de XML dans la présente spécification montrent l'utilisation du paramètre "charset" *(jeu de caractères)* dans l'en-tête Content-Type (défini dans la [RFC3023]) ainsi que les déclarations de jeu de caractères XML.

XML fournit aussi une capacité d'étiquetage de langage pour spécifier le langage des contenus d'un élément XML particulier. L'attribut "xml:lang" apparaît sur un élément XML pour identifier le langage de ses contenu et attributs. Voir dans [XML] les définitions des valeurs et de la portée.

Les applications WebDAV DOIVENT prendre en charge l'étiquetage du jeu de caractères, le codage du jeu de caractères, la fonction d'étiquetage du langage de la spécification XML. Les mises en œuvre d'applications WebDAV sont fortement invitées à lire dans "Types de supports XML" [RFC3023] les instructions sur le type de support MIME à utiliser pour le transport XML, et sur l'utilisation du paramètre charset de l'en-tête Content-Type.

Les noms utilisés dans la présente spécification entrent dans quatre catégories : noms des éléments de protocole comme les méthodes et en-têtes, noms des éléments XML, noms des propriétés, et noms des conditions. La dénomination des éléments de protocole suit celle de HTTP, utilisant des noms anglais codés en US-ASCII pour les méthodes et en-têtes. Comme ces éléments de protocole ne sont pas visibles aux utilisateurs, et sont simplement de longs identifiants de jetons, il n'ont pas besoin de prendre en charge plusieurs langages. De même, les noms des éléments XML utilisés dans la présente spécification ne sont pas visibles à l'utilisateur et n'ont donc pas besoin de prendre en charge plusieurs langages.

Les noms de propriété WebDAV sont des noms XML qualifiés (paires de nom d'espace de noms XML et de nom local). Bien que certaines applications (par exemple, un visionneur de propriété générique) affichent directement les noms de propriété à leurs utilisateurs, il est prévu que l'application normale utilise un ensemble fixe de propriétés, et fournisse une transposition du nom de propriété et de l'espace de noms en un champ lisible par l'homme quand elle affiche le nom de la propriété à un utilisateur. C'est seulement dans le cas où l'ensemble de propriétés n'est pas connu à l'avance qu'une application a besoin d'afficher un nom de propriété à un utilisateur. On recommande que les applications fournissent des noms de propriété lisibles par l'homme chaque fois que possible.

Pour le rapport des erreurs, on suit la convention des codes d'état de HTTP/1.1, incluant avec chaque code d'état une brève description du code (par exemple, 423 (Verrouillé)). Bien qu'existe la possibilité qu'un agent d'utilisateur peu doué affiche ce message à un utilisateur, les applications internationalisées vont ignorer ce message, et afficher un message approprié dans le langage et jeu de caractères de l'utilisateur.

Comme l'interopération des clients et des serveurs n'exige pas d'informations locales, la présente spécification ne spécifie pas de mécanisme pour la transmission de ces informations.

20. Considérations pour la sécurité

Cette Section est fournie pour préciser les questions concernant les implications de sécurité dont les applications WebDAV doivent avoir connaissance.

Toutes les considérations sur la sécurité de HTTP/1.1 (discutées dans la [RFC2616]) et de XML (discutées dans la [RFC3023]) s'appliquent aussi à WebDAV. De plus, les risques pour la sécurité inhérents à l'édition à distance exigent une technologie d'authentification plus forte, introduisent plusieurs nouveaux soucis de confidentialité, et peuvent augmenter les risques découlant d'une conception faible du serveur. Ces problèmes sont détaillés ci-dessous.

20.1 Authentification des clients

Du fait de leur accent sur l'édition, les serveurs WebDAV ont besoin d'utiliser une technique d'authentification pour protéger non seulement l'accès à une ressource du réseau, mais aussi l'intégrité de la ressource. De plus, l'introduction de la fonction de verrouillage exige la prise en charge de l'authentification.

Un mot de passe envoyé en clair sur un canal non sûr est un moyen inadéquat pour protéger l'accessibilité et l'intégrité d'une ressource car le mot de passe peut être intercepté. Comme l'authentification de base pour HTTP/1.1 effectue essentiellement une transmission en clair d'un mot de passe, l'authentification de base NE DOIT PAS être utilisée pour authentifier un client WebDAV auprès d'un serveur sauf si la connexion est sûre. De plus, un serveur WebDAV NE DOIT PAS envoyer un défi d'authentification de base dans un en-tête WWW-Authenticate sauf si la connexion est sûre. Un exemple de connexion sûre serait une connexion de sécurité de la couche Transport (TLS, *Transport Layer Security*) employant une suite de chiffrement forte et l'authentification du serveur.

Les applications WebDAV DOIVENT prendre en charge le schéma d'authentification par résumé [RFC2617]. Comme l'authentification par résumé vérifie que les deux parties à une communication connaissent un secret partagé, un mot de passe, sans avoir à envoyer ce secret en clair, l'authentification par résumé évite les problèmes de sécurité inhérents à l'authentification de base tout en fournissant un niveau d'authentification qui est utile dans une large gamme de scénarios.

20.2 Déni de service

Les attaques de déni de service sont un souci particulier des serveurs WebDA. WebDAV plus HTTP permet des attaques de déni de service sur toutes les parties des ressources d'un système.

- La mémorisation sous-jacente peut être attaquée en mettant des fichier extrêmement grands.
- Demander des opérations récurrentes sur de grandes collections peut attaquer le temps de traitement.
- Faire plusieurs demandes en parallèle sur plusieurs connexions peut attaquer les connexions au réseau.

Les serveurs WebDAV doivent avoir connaissance de la possibilité d'une attaque de déni de service à tous les niveaux. La réponse appropriée à une telle attaque PEUT être de simplement éliminer la connexion. Ou, si le serveur est capable de faire une réponse, le serveur PEUT utiliser une demande d'état de niveau 400 comme un 400 (Mauvaise demande) et indiquer pourquoi la demande est refusée (une réponse d'état de niveau 500 indiquerait que le problème est au serveur, tandis que les attaques de DoS non intentionnelles sont des choses auxquelles le client est capable de remédier).

20.3 Sécurité par l'obscurité

WebDAV fournit, par la méthode PROPFIND, un mécanisme pour faire la liste des ressources membres d'une collection. Cela diminue largement l'efficacité des techniques de sécurité ou de confidentialité qui s'appuient seulement sur la difficulté de découvrir les noms des ressources du réseau. Les utilisateurs de serveurs WebDAV sont invités à utiliser les techniques de contrôle d'accès pour prévenir l'accès non autorisé aux ressources, plutôt que de dépendre de la relative obscurité de leurs noms de ressources.

20.4 Questions de confidentialité en rapport avec les verrous

Quand il soumet une demande de verrou, un agent d'utilisateur peut aussi soumettre un champ XML 'owner' donnant des informations de contact pour la personne qui ôte le verrou (pour les cas où une personne, plutôt qu'un robot, ôte le verrou). Ces informations de contact sont mémorisées dans une propriété DAV: lockdiscovery sur la ressource, et peuvent être utilisées par d'autres collaborateurs pour commencer la négociation sur l'accès à la ressource. Cependant, dans de nombreux cas, ces informations de contact peuvent être très confidentielles, et ne devraient pas être largement disseminées. Les serveurs DEVRAIENT limiter l'accès en écriture à la propriété DAV:lockdiscovery comme approprié. De plus, les agents d'utilisateur DEVRAIENT fournir le contrôle sur si les informations de contact sont envoyées, et si elles le sont, le contrôle sur exactement quelles informations sont envoyées.

20.5 Questions de confidentialité en rapport avec les propriétés

Comme les valeurs de propriété sont normalement utilisées pour contenir des informations comme l'auteur d'un document, il y a une possibilité que des soucis de confidentialité puissent apparaître en raison d'un large accès aux données de propriété d'une ressource. Pour réduire le risque de divulgation involontaire d'informations confidentielles via les propriétés, les serveurs sont invités à développer des mécanismes de contrôle d'accès qui séparent l'accès en lecture au coprs de la ressource et l'accès en lecture aux propriétés de la ressource. Cela permet à un utilisateur de contrôler la

dissémination des données de propriété sans trop restreindre l'accès au contenu de la ressource.

20.6. Implications des entités XML

XML prend en charge une facilité appelée "entités externes", définie au paragraphe 4.2.2 de [XML], qui donne pour instruction à un processeur XML de restituer et inclure du XML supplémentaire. Une entité externe XML peut être utilisée pour ajouter ou modifier la déclaration de type de document (DTD, *document type declaration*) associée à un document XML. Une entité externe XML peut aussi être utilisée pour inclure du XML dans le contenu d'un document XML. Pour le XML non validant, comme le XML utilisé dans la présente spécification, l'inclusion d'une entité XML externe n'est pas exigé par XML. Cependant, XML déclare qu'un processeur XML peut, à sa discrétion, inclure l'entité externe XML.

Les entités externes XML n'ont pas de confiance inhérente et sont sujettes à toutes les attaques endémique de toute demande GET HTTP. De plus, il est possible à une entité externe XML de modifier le DTD, et donc d'affecter la forme finale d'un document XML, dans le pire des cas, modifiant significativement sa sémantique ou exposant le processeur XML aux risques exposés dans la [RFC3023]. Donc, les mises en œuvre doivent être conscientes que les entités externes XML devraient être traitées comme n'étant pas de confiance. Si un serveur choisit de ne pas traiter les entités externes XML, il DEVRAIT répondre aux demandes contenant des entités externes avec le code de condition 'no-external-entities'.

Il y a aussi le risque d'adaptabilité qui accompagnerait une application largement déployée qui fait usage d'entités externes XML. Dans cette situation, il est possible qu'il y ait un nombre significatif de demandes pour une entité externe XML, débordant potentiellement tout serveur qui fait des demandes pour la ressource qui contient l'entité externe XML.

De plus, il y a aussi un risque fondé sur l'évaluation des "entités internes" comme définies au paragraphe 4.2.2 de [XML]. Une petite demande, conçue avec soin en utilisant des entités internes incorporées peut exiger d'énormes quantités de mémoire et/ou de temps de traitement. Les mises en œuvre de serveur devraient être conscientes de ce risque et configurer leurs analyseurs XML de façon à ce que des demandes comme celles là puissent être détectées et rejetées aussitôt que possible.

20.7 Risques liés aux jetons de verrou

La présente spécification (paragraphe 6.5) encourage l'utilisation d'un "espace de noms d'URN d'identifiant univoque universel (UUID, *Universally Unique Identifier*)" [RFC4122] pour les jetons de verrou, afin de garantir leur unicité dans l'espace et le temps. Les UUID de version 1 (définis à la Section 4) PEUVENT contenir un champ "node" qui "consiste en une adresse MAC IEEE 802, généralement l'adresse de l'hôte. Pour les systèmes avec plusieurs adresses IEEE, toute adresse disponible peut être utilisée". Comme un serveur WebDAV va produire de nombreux verrous pendant sa durée de vie, cela implique qu'il peut aussi exposer publiquement son adresse IEEE 802.

Plusieurs risques sont associés à l'exposition des adresses IEEE 802. En utilisant l'adresse IEEE 802 :
- Il est possible de retracer les mouvements d'un matériel de sous réseau en sous réseau.
- Il est possible d'identifier le fabricant du matériel d'un serveur WebDAV.
- Il est possible de déterminer le nombre de chaque type d'ordinateur qui fonctionne avec WebDAV.

Ce risque s'applique seulement aux versions d'UUID fondées sur l'adresse d'hôte. La Section 4 de la [RFC4122] décrit plusieurs autres mécanismes pour générer des UUID qui n'impliquent pas l'adresse d'hôte et ne subissent donc pas ce risque.

20.8 Hébergement de contenu malveillant

HTTP a la capacité d'héberger des programmes qui sont exécutés sur des machines de clients. Ces programmes peuvent prendre de nombreuses formes incluant des scénarios sur la Toile, des exécutables, des modules prêts à brancher, et des macros dans des documents. WebDAV ne change aucun des problèmes de sécurité pour ces programmes, car souvent WebDAV est utilisé dans des contextes où une large gamme d'utilisateurs peut publier des documents sur un serveur. Le serveur pourrait ne pas avoir une relation de confiance étroite avec l'auteur qui publie le document. Les serveurs qui permettent aux clients de publier un contenu arbitraire peuvent utilement mettre en œuvre des précautions pour vérifier que le contenu publié au serveur n'est pas dommageable pour d'autres clients. Les serveurs pourraient faire cela par des techniques comme la restriction des types de contenu qui sont permis à la publication et appliquant un logiciel de détection de virus et malgiciels sur le contenu publié. Les serveurs peuvent aussi atténuer le risque en ayant des restrictions d'accès appropriées et par l'authentification des utilisateurs à qui il est permis de publier du contenu sur le serveur.

21. Considérations relatives à l'IANA

21.1 Nouveaux schémas d'URI

La présente spécification définit deux schémas d'URI :
1. le schéma "opaquelocktoken" défini à l'Appendice C, et
2. le schéma d'URI "DAV", qui a historiquement été utilisé dans la [RFC2518] pour préciser les noms de propriétés WebDAV et d'éléments XML et qui continue d'être utilisé à cette fin dans la présente spécification et d'autres qui étendent WebDAV. La création d'identifiants dans l'espace de noms "DAV:" est contrôlée par l'IETF.

Noter que la définition de nouveaux schémas d'URI pour les espaces de noms XML est maintenant déconseillée. "DAV:" a été défini avant qu'apparaissent les bonnes pratiques standard.

21.2 Espaces de noms XML

Les espaces de noms XML précisent les noms de propriétés WebDAV et d'éléments XML. Tout utilisateur ou application WebDAV peut définir un nouvel espace de noms afin de créer des propriétés personnalisées ou étendre la syntaxe XML de WebDAV. L'IANA n'a pas besoin de gérer de tels espaces de noms, noms de propriétés, ou noms d'éléments.

21.3 Champs d'en-tête de message

Les champs d'en-tête de message ci-dessous devraient être ajoutés au registre permanent (voir la [RFC3864]).

21.3.1 DAV

Nom de champ d'en-tête : DAV
Protocole applicable : http
Statut : standard
Auteur/Contrôleur des changements : IETF
Document de spécification : la présente spécification (paragraphe 10.1)

21.3.2 Depth

Nom de champ d'en-tête : Depth
Protocole applicable : http
Statut : standard
Auteur/Contrôleur des changements : IETF
Document de spécification : la présente spécification (paragraphe 10.2)

21.3.3 Destination

Nom de champ d'en-tête : Destination
Protocole applicable : http
Statut : standard
Auteur/Contrôleur des changements : IETF
Document de spécification : la présente spécification (paragraphe 10.3)

21.3.4 If

Nom de champ d'en-tête : If
Protocole applicable : http
Statut : standard
Auteur/Contrôleur des changements : IETF
Document de spécification : la présente spécification (paragraphe 10.4)

21.3.5 Lock-Token

Nom de champ d'en-tête : Lock-Token
Protocole applicable : http
Statut : standard
Auteur/Contrôleur des changements : IETF
Document de spécification : la présente spécification (paragraphe 10.5)

21.3.6 Overwrite

Nom de champ d'en-tête : Overwrite
Protocole applicable : http
Statut : standard
Auteur/Contrôleur des changements : IETF
Document de spécification : la présente spécification (paragraphe 10.6)

21.3.7 Timeout

Nom de champ d'en-tête : Timeout
Protocole applicable : http
Statut : standard
Auteur/Contrôleur des changements : IETF
Document de spécification : la présente spécification (paragraphe 10.7)

21.4 Codes d'état HTTP

La présente spécification définit les codes d'état HTTP
- o 207 Multi-états (paragraphe 11.1)
- o 422 Entité non traitable (paragraphe 11.2),
- o 423 Verrouillé (paragraphe 11.3),
- o 424 Échec de dépendance (paragraphe 11.4) et
- o 507 Mémoire insuffisante (paragraphe 11.5),

à mettre à jour dans le registre < http://www.iana.org/assignments/http-status-codes >.

Note : Le code d'état HTTP 102 (Traitement) a été supprimé dans la présente spécification ; son enregistrement IANA devrait continuer de référencer la RFC 2518.

22. Remerciements

Une spécification comme celle-ci prospère sur les revues critiques perçantes et se dessèche dans une négligence apathiquc. Les auteurs temoignent de leur reconnaissance aux contributeurs suivants, dont les apports ont été si précieux à chaque étape de notre travail.

Contributeurs à la RFC 2518
Terry Allen, Harald Alvestrand, Jim Amsden, Becky Anderson, Alan Babich, Sanford Barr, Dylan Barrell, Bernard Chester, Tim Berners-Lee, Dan Connolly, Jim Cunningham, Ron Daniel, Jr., Jim Davis, Keith Dawson, Mark Day, Brian Deen, Martin Duerst, David Durand, Lee Farrell, Chuck Fay, Wesley Felter, Roy Fielding, Mark Fisher, Alan Freier, George Florentine, Jim Gettys, Phill Hallam-Baker, Dennis Hamilton, Steve Henning, Mead Himelstein, Alex Hopmann, Andre van der Hoek, Ben Laurie, Paul Leach, Ora Lassila, Karen MacArthur, Steven Martin, Larry Masinter, Michael Mealling, Keith Moore, Thomas Narten, Henrik Nielsen, Kenji Ota, Bob Parker, Glenn Peterson, Jon Radoff, Saveen Reddy, Henry Sanders, Christopher Seiwald, Judith Slein, Mike Spreitzer, Einar Stefferud, Greg Stein, Ralph Swick, Kenji Takahashi, Richard N. Taylor, Robert Thau, John Turner, Sankar Virdhagriswaran, Fabio Vitali, Gregory Woodhouse, et Lauren Wood.

Dans cette liste deux personnes méritent une mention particulière. Les contributions de Larry Masinter ont été très précieuses ; il a aidé à la formation du groupe de travail et patiemment encadré les auteurs tout le long du chemin. De nombreuses façons, il a établi les standards élevés que nous nous sommes efforcés d'atteindre. Les contributions de Judith Slein ont aussi été précieuses ; en précisant les exigences et en revoyant patiemment version après version, elle a à la fois amélioré la présente spécification et enrichi nos vues sur la gestion de document.

Merci aussi à John Turner qui a développé le DTD XML.

Les auteurs de la RFC 2518 sont Yaron Goland, Jim Whitehead, A. Faizi, Steve Carter, et D. Jensen. Bien que leurs noms aient dû être retirés de la liste des auteurs à cause de restrictions de l'IETF sur le nombre des auteurs, ils peuvent être crédités de la majorité des concepts de WebDAV.

Remerciements supplémentaires pour la présente spécification

Les contributeurs significatifs au texte de la présente spécification sont mentionnés comme contributeurs à la Section suivante. On doit aussi remercier Geoff Clemm, Joel Soderberg, et Dan Brotsky pour le hachage du texte spécifique sur la

liste de diffusion ou dans les réunions. Joe Hildebrand et Cullen Jennings ont aidé à résoudre de nombreux problèmes. Barry Lind a décrit une considération de sécurité supplémentaire et Cullen Jennings a fourni du texte pour cette considération. Jason Crawford a suivi l'état des problèmes du présent document pendant des années, suivi par Elias Sinderson.

23. Contributeurs à la présente spécification

Julian Reschke
<green/>bytes GmbH
Hafenweg 16, 48155 Muenster,
Germany
mél : julian.reschke@greenbytes.de

Elias Sinderson
University of California, Santa Cruz
1156 High Street, Santa Cruz, CA
95064
mél : elias@cse.ucsc.edu

Jim Whitehead
University of California, Santa Cruz
1156 High Street, Santa Cruz, CA
95064
mél : ejw@soe.ucsc.edu

24. Auteurs de la RFC 2518

Y. Y. Goland
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052-6399
mél : yarong@microsoft.com

E. J. Whitehead, Jr.
Dept. Of Information et Computer Science
University of California, Irvine
Irvine, CA 92697-3425
mél : ejw@ics.uci.edu

A. Faizi
Netscape
685 East Middlefield Road
Mountain View, CA 94043
mél : asad@netscape.com

S. R. Carter
Novell
1555 N. Technology Way
M/S ORM F111
Orem, UT 84097-2399
mél : srcarter@novell.com

D. Jensen
Novell
1555 N. Technology Way
M/S ORM F111
Orem, UT 84097-2399
mél : dcjensen@novell.com

25. Références

25.1 Références normatives

[XML] Bray, T., Paoli, J., Sperberg-McQueen, C., Maler, E., and F. Yergeau, "Extensible Markup Language (XML) 1.0 (Fourth Edition)", W3C REC-xml-20060816, août 2006, <http://www.w3.org/TR/2006/REC-xml-20060816/ >.

[XML-INFOSET] Cowan, J. and R. Tobin, "XML Information Set (Second Edition)", W3C REC-xml-infoset-20040204, février 2004, < http://www.w3.org/TR/2004/REC-xml-infoset-20040204/ >.

[XML-NAMES] Bray, T., Hollander, D., Layman, A., and R. Tobin, "Namespaces in XML 1.0 (Second Edition)", W3C REC-xml-names-20060816, août 2006, < http://www.w3.org/TR/2006/REC-xml-names-20060816/ >.

[RFC2119] S. Bradner, "Mots clés à utiliser dans les RFC pour indiquer les niveaux d'exigence", BCP 14, mars 1997.

[RFC2277] H. Alvestrand, "Politique de l'IETF en matière de jeux de caractères et de langages", BCP 18, janvier 1998.

[RFC2616] R. Fielding et autres, "Protocole de transfert hypertexte -- HTTP/1.1", juin 1999. *(D.S., MàJ par* 2817, 6585)

[RFC2617] J. Franks et autres, "Authentification HTTP : Authentification d'accès de base et par résumé", RFC 2617, juin 1999. *(DS.)*

[RFC3339] G. Klyne, C. Newman, "La date et l'heure sur l'Internet : horodatages", juillet 2002. *(P.S.)*

[RFC3629] F. Yergeau, "UTF-8, un format de transformation de la norme ISO 10646", STD 63, novembre 2003.

[RFC3986] T. Berners-Lee, R. Fielding et L. Masinter, "Identifiant de ressource uniforme (URI) : Syntaxe générique", STD 66, janvier 2005.

[RFC4122] P. Leach et autres, "Espace de noms d'URN d'identifiant univoque universel (UUID)", juillet 2005. *(P.S.)*

25.2 Références pour information

[RFC2291] J. Slein, F. Vitali, E. Whitehead, D. Durand, "Exigences pour un protocole réparti des noms d'auteurs et des numéros de version pour la Toile mondiale", février 1998. *(Information)*

[RFC2518] Y. Goland, E. Whitehead, A. Faizi, S. Carter et D. Jensen, "Extensions HTTP pour la création répartie -- WEBDAV", février 1999. *(Obsolète, voir la* RFC4918)

[RFC2781] P. Hoffman et F. Yergeau, "UTF-16, un codage de la norme ISO 10646", février 2000.

[RFC3023] M. Murata, S. St.Laurent et D. Kohn, "Types de support XML", janvier 2001. (*Obsolète, voir* RFC7303)

[RFC3253] G. Clemm et autres, "Extensions de versions à WebDAV (Protocole de collecte ordonnée des auteurs et des versions distribuée sur la Toile)", mars 2002. *(P.S.)*

[RFC3648] J. Whitehead, J. Reschke, éd., "Protocole de collecte ordonnée des auteurs et des versions distribuée sur la Toile (WebDAV)", décembre 2003. *(P.S.)*

[RFC3744] G. Clemm et autres, "Protocole de contrôle d'accès à la collecte des noms d'auteurs et de version répartie sur la Toile (WebDAV)", mai 2004. *(P.S.)*

[RFC3864] G. Klyne, M. Nottingham, J. Mogul, "Procédures d'enregistrement pour les champs d'en-tête de message", septembre 2004. (BCP0090)

Appendice A. Notes sur le traitement des éléments XML

A.1 Notes sur les éléments XML vides

XML prend en charge deux mécanismes pour indiquer qu'un élément XML n'a aucun contenu. Le premier est de déclarer un élément XML de la forme <A></A>. Le second est de déclarer un élément XML de la forme <A/>. Les deux éléments XML sont sémantiquement identiques.

A.2 Notes sur le traitement XML illégal

XML est un format de données souple qui rend facile de soumettre des données qui paraissent légales mais en fait ne le sont pas. La philosophie de "être souple dans ce qu'on accepte et strict dans ce qu'on envoie" s'applique toujours, mais elle ne doit pas être appliquée à tort. XML est extrêmement souple pour traiter des problèmes d'espaces, d'ordre des éléments, d'insertion de nouveaux éléments, etc. Cette souplesse n'exige pas d'extension, en particulier pas dans le domaine de la signification des éléments.

Il n'y a pas de "gentillesse" à accepter une combinaison illégale d'éléments XML. Au mieux, cela va causer un résultat non voulu et au pire, cela peut causer des dommages réels.

A.3 Exemple – erreur de syntaxe XML

Le corps de demande suivant pour une méthode PROPFIND est illégal.

```
<?xml version="1.0" encoding="utf-8" ?>
<D:propfind xmlns:D="DAV:">
 <D:allprop/>
 <D:propname/>
</D:propfind>
```

La définition de l'élément propfind permet seulement l'élément allprop ou l'élément propname, pas les deux. Donc, c'est une erreur à laquelle on doit répondre avec un code 400 (Mauvaise demande).

Imaginons cependant qu'un serveur veuille être "gentil" et décide de prendre l'élément allprop comme le vrai élément et d'y

répondre. Un client qui fonctionne sur une ligne avec une bande passante limitée et qui a l'intention d'exécuter un propname serait très surpris si le serveur traite la commande comme un allprop.

De plus, si un serveur est négligent et décide de repondre à cette demande, le résultat va varier de façon aléatoire d'un serveur à l'autre, certains exécutant la directive allprop, et d'autres exécutant la directive propname. Cela réduit plutôt qu'augmente l'interopérabilité.

A.4 Exemple - élément XML inattendu

L'exemple précédent était illégal parce que il contenait deux éléments à qui il est explicitement interdit d'apparaître ensemble dans l'élément propfind. Cependant, XML est un langage extensible, de sorte qu'on peut imaginer de nouveaux éléments définis pour être utilisés avec propfind. Voici le corps de la demande d'un PROPFIND et, comme dans l'exemple précédent, il doit être rejeté avec un 400 (Mauvaise demande) par un serveur qui ne comprend pas l'élément expired-props.

```
<?xml version="1.0" encoding="utf-8" ?>
<D:propfind xmlns:D="DAV:"xmlns:E="http://www.exemple.com/standards/props/"><E:expired-props/>
</D:propfind>
```

Pour comprendre pourquoi un 400 (Mauvaise demande) est retourné, regardons le corps de demande comme le serveur qui n'est pas familier de expired-props le voit.

```
<?xml version="1.0" encoding="utf-8" ?>
<D:propfind xmlns:D="DAV:"xmlns:E="http://www.exemple.com/standards/props/">
</D:propfind>
```

Comme le serveur ne comprend pas l'élément 'expired-props', conformément aux règles de traitement XML spécifiques de WebDAV spécifiées à la Section 17, il doit traiter la demande comme si l'élément n'était pas là. Donc, le serveur voit un propfind vide, qui selon la définition de l'élément propfind est illégal.

Noter que l'extension étant additive, elle n'aurait pas nécessairement résulté en un 400 (Mauvaise demande). Par exemple, imaginons le corps de demande suivant pour un PROPFIND :

```
<?xml version="1.0" encoding="utf-8" ?>
<D:propfind xmlns:D="DAV:"xmlns:E="http://www.exemple.com/standards/props/">
 <D:propname/>
 <E:leave-out>*boss*</E:leave-out>
</D:propfind>
```

L'exemple précédent contient l'élément fictif "leave-out". Son objet est d'empêcher le retour de toute propriété dont le nom correspond au schéma soumis. Si l'exemple précédent était soumis à un serveur qui ne connaît pas "leave-out", le seul résultat serait que l'élément "leave-out" serait ignoré et qu'un propname serait exécuté.

Appendice B. Notes sur la compatibilité client HTTP

WebDAV a été conçu pour être, et est effectivement, rétro-compatible avec HTTP 1.1. Les méthodes PUT et DELETE sont définies dans HTTP et peuvent donc être utilisées par les clients HTTP ainsi que les clients à capacité WebDAV, mais les réponses à PUT et DELETE ont été étendues dans la présente spécification de façon que seulement un client WebDAV y soit entièrement prêt. Des problèmes théoriques ont été soulevés sur la question de savoir si ces réponses causeraient des problèmes d'interopérabilité avec les clients seulement HTTP, et cette section vise ces problèmes.

Comme tout client HTTP devrait traiter les codes d’état non reconnus de niveau 400 et 500 comme des erreurs, les nouveaux codes d’état suivants ne devraient pas poser de problèmes : 422, 423, et 507 (424 est aussi un nouveau code d'état mais il apparaît seulement dans le corps d'une réponse multi états). Donc, par exemple, si un client HTTP tente un PUT ou DELETE sur une ressource verrouillée, la réponse 423 Verrouillée devrait résulter en une erreur générique présentée à l'utilisateur.

La réponse 207 Multi états est intéressante parce que un client HTTP qui produit une demande DELETE sur une collection pourrait interpréter une réponse 207 comme un succès, même si il ne réalise pas que la ressource est une collection et ne peut pas comprendre que l'opération DELETE pourrait avoir été un échec complet ou partiel. Cette interprétation n'est pas entièrement justifiée, parce que une réponse de niveau 200 indique que le serveur "a reçu, compris, et accepté" la demande,

et non que la demande a résulté en un succès complet.

Une option est qu'un serveur pourrait traiter un DELETE d'une collection comme une opération atomique, et utiliser soit 204 (Pas de contenu) en cas de succès, ou une réponse d'erreur appropriée (de niveau 400 ou 500) pour une erreur. Cette approche maximiserait bien sûr la rétro compatibilité. Cependant, comme les essais d'interopérabilité et les discussions du groupe de travail n'ont pas donné d'instances de clients HTTP produisant une demande DELETE contre une collection WebDAV, ce problème est plus théorique que pratique. Donc, les serveurs vont probablement réussir complètement à interopérer avec les clients HTTP même si ils traitent toute demande DELETE de collection comme une demande WebDAV et envoient une réponse 207 Multi-états.

En général, les mises en œuvre de serveur sont invitées à utiliser les réponses détaillées et les autres mécanismes définis dans le présent document plutôt que de faire des changements pour des soucis théoriques d'interopérabilité.

Appendice C. Schémas et URI 'opaquelocktoken'

Le schéma d'URI 'opaquelocktoken' a été défini dans la [RFC2518] (et enregistré par l'IANA) afin de créer des URI syntaxiquement corrects et faciles à générer à partir des UUID, destinés à être utilisés comme des jetons de verrous et à être uniques tout le temps sur toutes les ressources.

Un URI opaquelocktoken est construit en enchaînant le schéma 'opaquelocktoken' avec un UUID, ainsi qu'une extension facultative. Les serveurs peuvent créer de nouveaux UUID pour chaque nouveau jeton de verrou. Si un serveur souhaite réutiliser les UUID, le serveur DOIT ajouter une extension, et l'algorithme qui génère l'extension DOIT garantir que la même extension ne va jamais être utilisée deux fois avec l'UUID associé.

OpaqueLockToken-URI = "opaquelocktoken:" UUID [Extension]
; UUID est défini à la Section 3 de la [RFC4122]. Noter que les LWS ne sont pas permises entre les éléments de cette production.

Extension = path
; path est défini au paragraphe 3.3 de la [RFC3986]

Appendice D. Ressources Lock-null

Le modèle WebDAV original pour le verrouillage des URL non transposés a créé des "ressources lock-null". Ce modèle était trop compliqué et des problèmes d'interopérabilité et de mise en œuvre ont été découverts. Le nouveau modèle WebDAV pour le verrouillage des URL non transposés (voir au paragraphe 7.3) crée des "ressources vides verrouillées". Les ressources Lock-null sont déconseillées. Cette section discute brièvement du modèle original parce que les clients DOIVENT être capables de traiter l'un et l'autre modèle.

Dans le modèle original "ressource lock-null", qui n'est plus recommandé pour la mise en œuvre :

- une ressource lock-null apparaît parfois comme "Non trouvée". Le serveur répond avec un 404 ou 405 à toute méthode sauf pour PUT, MKCOL, OPTIONS, PROPFIND, LOCK, UNLOCK ;
- une ressource lock-null n'apparaît cependant pas comme un membre de sa collection parente ;
- le serveur supprime entièrement la ressource lock-null (son URI devient non transposé) si son verrou s'en va avant qu'elle soit convertie en une ressource régulière. On rappelle que les verrous s'en vont non seulement quand ils expirent ou sont déverrouillés, mais sont aussi supprimés si une ressource est débaptisée ou déplacée, ou si une collection parente est débaptisée ou déplacée ;
- le serveur convertit la ressource lock-null en une ressource régulière si une demande PUT à l'URL réussit ;
- le serveur convertit la ressource lock-null en une collection si une demande MKCOL à l'URL réussit (bien que l'expérience de l'interopérabilité ait montré que tous les serveurs ne respectent pas cette exigence) ;
- des valeurs de propriétés ont été définies pour les propriétés DAV:lockdiscovery et DAV:supportedlock mais pas nécessairement pour les autres propriétés comme DAV:getcontenttype.

Les clients peuvent facilement interopérer avec les serveurs qui prennent en charge le vieux modèle de "ressource lock-null" et le modèle recommandé de "ressource verrouillée vide" en tentant seulement un PUT après un LOCK sur un URL non transposé, et pas un MKCOL ou GET.

D.1 Lignes directrices pour les clients qui utilisent LOCK pour créer des ressources

Un client WebDAV qui met en œuvre la présente spécification pourrait trouver des serveurs qui créent des ressources lock-null (mises en œuvre avant la présente spécification en utilisant la [RFC2518]) ainsi que des serveurs qui créent des ressources vides verrouillées. La réponse à la demande LOCK ne va pas indiquer quelle sorte de ressource a été créée. Il existe quelques techniques qui aident le client à traiter l'un et l'autre type.

Si le client souhaite éviter de créer accidentellement soit des ressources lock-null, soit des ressources vides verrouillées, un en-tête "If-Match: *" peut être inclus avec les demandes LOCK pour empêcher le serveur de créer une nouvelle ressource.

Si une demande LOCK crée une ressource et si le client veut ensuite écraser cette ressource en utilisant une demande COPY ou MOVE, le client devrait inclure un en-tête "Overwrite: T".

Si une demande LOCK crée une ressource et si le client décide alors de se débarasser de cette ressource, une demande DELETE est supposée échouer sur une ressource lock-null et UNLOCK devrait être utilisé à la place. Mais avec une ressource vide verrouillée, UNLOCK ne fait pas disparaître la ressource. Donc, le client pourrait devoir essayer les deux demandes et ignorer une erreur dans une des deux demandes.

Appendix E. Lignes directrices pour les clients qui désirent s’authentifier

De nombreux clients WebDAV qui ont déjà été mis en œuvre ont des réglages de comptes (de la même façon que les clients de messagerie électronique mémorisent leurs réglages de compte IMAP). Donc, le client WebDAV serait capable de s'authentifier avec son premier couple de demandes auprès du serveur, pourvu qu'il ait un moyen d'obtenir le défi d'authentification provenant du serveur avec le nom de domaine, le nom occasionnel, et les autres informations du défi. Noter que les résultats de certaines demandes pourraient varier selon que le client est authentifié ou non -- un PROPFIND pourrait retourner des ressources plus visibles si le client est authentifié, et ne pas échouer si le client est anonyme.

Il y a un certain nombre de façons pour que le client soit capable de déclencher chez le serveur la fourniture d'un défi d'authentification. Le présent appendice décrit un couple d'approches qui semblent avoir de grandes chances de donner de bons résultats.

La première approche est d'effectuer une demande qui devrait exiger l'authentification. Cependant, il est possible qu'un serveur puisse traiter toute demande même sans authentification, donc, pour être entièrement sûr, le client pourrait ajouter un en-tête conditionnel pour s'assurer que même si la demande passe les vérifications de permission, elle ne soit en fait pas traitée par le serveur. Un exemple de cette approche serait d'utiliser une demande PUT avec un en-tête "If-Match" avec la valeur ETag constituée. Cette approche pourrait échouer à produire un défi d'authentification si le serveur ne vérifie pas l'autorisation avant les essais conditionnels comme requis (voir au paragraphe 8.5) ou si le serveur n'a pas besoin de vérifier l'autorisation.

Exemple - forcer le défi d'authentification avec demande d'écriture

>>Demande

```
PUT /forceauth.txt HTTP/1.1
Host: www.exemple.com
If-Match: "xxx"
Content-Type: text/plain
Content-Length: 0
```

La seconde approche est d'utiliser un en-tête Authorization (défini dans la [RFC2617]) qui va probablement être rejeté par le serveur mais qui va alors inviter à un défi d'authentification approprié. Par exemple, le client pourrait commencer par une demande PROPFIND contenant un en-tête Authorization contenant une chaîne de base userid:password, ou avec des accréditifs réels plausibles. Cette approche s'appuie sur une réponse du serveur avec un "401 Non autorisé" ainsi qu'un défi si il reçoit un en-tête Authorization avec un nom d'utilisateur non reconnu, un mot de passe invalide, ou si il ne prend même pas en charge l'authentification de base. Cela semble devoir fonctionner à cause des exigences de la RFC 2617 : "Si le serveur d'origine ne souhaite pas accepter les accréditifs envoyés avec une demande, il DEVRAIT retourner une réponse 401 (Non autorisée). La réponse DOIT inclure un champ d'en-tête WWW-Authenticate contenant au moins un défi (éventuellement nouveau) applicable à la ressource demandée."

Il y a un petit problème de mise en œuvre de cette recommandation dans certains cas, parce que certains serveurs n'ont même pas les informations de défi pour certaines ressources. Donc, quand il n'y a pas de moyen de s'authentifier pour une

ressource ou quand la ressource est entièrement disponible au public sur toutes les méthodes acceptées, le serveur PEUT ignorer l'en-tête Authorization, et le client va probablement réessayer plus tard.

Exemple - forcer le défi d'authentification avec l'en-tête Authorization :

>>Demande

```
PROPFIND /docs/ HTTP/1.1
Host: www.exemple.com
Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==
Content-type: application/xml; charset="utf-8"
Content-Length: xxxx

[corps omis]
```

Appendice F. Résumé des changements par rapport à la RFC 2518

Cette Section fait la liste des changements majeurs entre le présent document et la RFC 2518, en commençant par ceux qui vont probablement résulter en des changements de mise en œuvre. Les serveurs vont annoncer la prise en charge de tous les changements dans la présente spécification en retournant la classe de conformité "3" dans l'en-tête de réponse DAV (voir les paragraphes 10.1 et 18.3).

F.1 Changements pour les mises en œuvre de client et de serveur

Opérations de collections et d'espace de noms

- La sémantique de PROPFIND 'allprop' (paragaphe 9.1) a été assouplie afin que les serveurs retournent des résultats incluant, au minimum, les propriétés vives définies dans la présente spécification, mais pas nécessairement d'autres propriétés vives. La directive 'allprop' signifie donc quelque chose comme "retourner toutes les propriétés qui sont supposées être retournées quand 'allprop' est demandé" -- un ensemble de propriétés qui peut inclure des propriétés personnalisées et des propriétés définies dans d'autres spécifications si ces autres spécifications l'exigent. Par rapport à cela, les demandes 'allprop' peuvent maintenant être étendues avec la syntaxe 'include' pour inclure des propriétés désignées spécifiques, évitant ainsi des demandes supplémentaires dues au changement de sémantique de 'allprop'.
- Il est maintenant permis aux serveurs de rejeter les demandes PROPFIND avec Depth: Infinity. Les clients qui utilisaient cela vont devoir à la place être capables de faire une série de demandes Depth:1.
- Les corps de réponse Multi-états peuvent maintenant transporter la valeur de l'en-tête de réponse Location de HTTP dans le nouvel élément 'location'. Les clients peuvent utiliser cela pour éviter des allers-retours supplémentaires au serveur quand il y a un élément 'response' avec un état 3xx (voir au paragraphe 14.24).
- La définition de COPY a été assouplie afin de ne plus exiger des serveurs qu'ils suppriment d'abord les ressources cibles (c'était une incompatibilité connue avec la [RFC3253]). Voir au paragraphe 9.8.

En-têtes et mise en ordre

- Les en-têtes de demande Destination et If permettent maintenant des chemins absolus en plus des URI complets (voir au paragraphe 8.3). Cela peut être utile aux clients qui opèrent à travers un mandataire inverse qui réécrit l'en-tête Hôte demandé, mais pas les en-têtes spécifiques de WebDAV.
- La présente spécification adopte les extensions de mise en ordre des erreurs et la terminologie "précondition/postcondition" définie dans la [RFC3253] (voir la Section 16). En rapport avec cela, elle ajoute l'élément XML "error" dans le corps de réponse multi états (voir au paragraphe 14.5, cependant noter qu'elle utilise un format différent de celui recommandé dans la RFC 3253).
- Les envoyeurs et receveurs sont maintenant obligés de prendre en charge le codage de caractères UTF-16 dans les corps de messages XML (voir la Section 19).
- Les clients sont maintenat obligés d'envoyer l'en-tête Depth sur les demandes PROPFIND, bien que les serveurs soient toujours invités à prendre en charge les clients qui ne le font pas.

Verrouillage

- Le concept de la RFC 2518 de "ressource nulle verrouillée" (LNR, *lock-null ressource*) a été remplacé par une approche simplifiée, celui de "ressource verrouillée vide" (voir au paragraphe 7.3). Certains aspects des ressources lock-null ne peuvent plus être utilisés par les clients, à savoir, la capacité de les utiliser pour créer une collection verrouillée ou le fait qu'elles disparaissent sur un UNLOCK quand aucune demande PUT ou MKCOL n'a été produite. Noter qu'il est toujours permis aux serveurs de mettre en œuvre les LNR selon la RFC 2518.
- Il n'y a plus de rafraîchissement implicite des verrous. Les verrous sont seulement rafraîchis sur demande explicite (voir

au paragraphe 9.10.2).
- On a précisé que la valeur DAV:owner fournie dans la demande LOCK doit être préservée par le serveur tout comme une propriété morte (paragraphe 14.17). On a aussi ajouté l'élément DAV:lockroot (paragraphe 14.12) qui permet aux clients de découvrir la racine d'un verrou.

F.2 Changements pour les mises en œuvre de serveur

Opérations de collections et d'espace de noms
- Du fait de problèmes d'interopérabilité, les formats de contenus admis d'éléments 'href' dans les réponses multi-états ont été limités (voir au paragraphe 8.3).
- Du fait du manque de mise en œuvre, la prise en charge du corps de demande 'propertybehavior' pour COPY et MOVE a été supprimé. À la place, les exigences de préservation de propriété ont été précisées (voir les paragraphes 9.8 et 9.9).

Propriétés
- On a renforcé les exigences de serveur pour la mémorisation des valeurs de propriété, en particulier la persistance des informations de langage (xml:lang), les espaces, et les informations d'espace de noms XML (voir au paragraphe 4.3).
- On a précisé les exigences sur quelles propriétés devraient être écrivables par le client ; en particulier, le réglage de "DAV:displayname" devrait être pris en charge par les serveurs (voir la Section 15).
- Seules les productions 'rfc1123-date' sont légales comme valeurs pour DAV: getlastmodified (voir au paragraphe 15.7).

En-têtes et mise en ordre
- Les serveurs sont maintenant obligés de vérifier les autorisations avant de traiter des en-têtes conditionnels (voir au paragraphe 8.5).

Verrouillage
- Renforcement de l'exigence de vérification de l'identité du créateur de verrou lors de l'accès à des ressources verrouillées (voir au paragraphe 6.4). Les clients devraient savoir que les jetons de verrou retournés aux autres principaux peuvent seulement être utilisés pour ouvrir un verrou, si il en est.
- Le paragraphe 8.10.4 de la [RFC2518] exigeait à tort des serveurs qu'ils retournent un état 409 alors qu'un état 207 est en réalité approprié. Cela a été corrigé (paragraphe 9.10).

F.3 Autres changements

La définition de l'état de collection a été corrigé de façon qu'elle ne varie plus selon l'URI de demande (paragraphe 5.2).

La propriété DAV:source introduite au paragraphe 4.6 de la [RFC2518] a été supprimée à cause du manque d'expérience de mise en œuvre.

L'en-tête DAV permet maintenant des extensions non IETF par des URI en plus de la conformité aux jetons de classe. Il peut aussi être maintenant utilisé dans les demandes, bien que la présente spécification ne définisse aucune sémantique associée aux classes de conformité définies ici (paragraphe 10.1).

Dans la RFC 2518, la définition de l'en-tête Depth (paragraphe 9.2) exigeait que, par défaut, les en-têtes de demande seraient appliquées à la portée de chaque ressource. Sur la base de l'expérience de mise en œuvre, la valeur par défaut a maintenant été inversée (paragraphe 10.2).

Les définitions de code d'état HTTP 102 ([RFC2518], paragraphe 10.1) et l'en-tête de réponse Status-URI (paragraphe 9.7) ont été supprimées pour leur absence de mise en œuvre.

Le format TimeType utilisé dans l'en-tête de demande Timeout et l'élément XML "timeout" utilisé pour être extensible. Maintenant, seuls les deux formats définis par la présente spécification sont permis (voir au paragraphe 10.7).

Adresse de l'éditeur

Lisa Dusseault (éditeur)
CommerceNet
2064 Edgewood Dr.
Palo Alto, CA 94303
USA
mél : ldusseault@commerce.net

Notice complète de droits de reproduction

Remerciement
Le financement de la fonction d'édition des RFC est actuellement fourni par la Internet Society.