| | |
|---|---|
| Groupe de travail Réseau | Y. Goland, M |
| **RFC2518** | E J. Whitehe |
| Catégorie : En cours de normalisation | A Faizi, Ne |
| Rendue obsolète par la RFC4918 | D Jensen, N |
| Traduction Jean-Jacques Thomasson : <jjthomasson@nagora.com> | publiée en févrie |

# Extensions de HTTP pour la rédaction distribuée (WEBDAV)

---



Résumé :

Ce document contient les spécifications d'un ensemble de méthodes, en-têtes, et de types de contenus auxiliaires à HTTP/1.1 pour la gestion des propriétés des ressources, la création et la gestion de collections de ressources, la manipulation d'espaces de noms, et le verrouillage des ressources (pour éviter les collisions).

---

## Table des matières

## 1. Introduction

Ce document décrit une extension du protocole HTTP/1.1 qui permet aux clients WEB de participer à des processus éditoriaux à distance. Cette extension fournit un ensemble cohérent de méthodes, en-têtes, formats de corps de requêtes et de réponses qui permettent de faire les opérations suivantes :

- En ce qui concerne les **propriétés** : pouvoir créer, supprimer et obtenir des informations sur

des pages WEB, tel que le nom de leurs auteurs, leurs dates de création, et pouvoir lier des pages entre elles, quel qu'en soit le format.

- En ce qui concerne les **collections** : pouvoir créer des ensembles de documents et obtenir une liste hiérarchique des membres (à la manière d'une arborescence de répertoires et de fichiers).

- En ce qui concerne le **verrouillage** : pouvoir garantir l'unicité d'accès à un document (pas plus d'une personne au même moment) pour éviter le problème de "dernière modification perdue", qui survient quand des auteurs successifs travaillent sur des copies d'un même document sans fusionner à la fin leurs modifications respectives.

- Les opérations sur les **espaces de noms** : La possibilité de dire au serveur de copier ou déplacer des ressources WEB.

Le raisonnement et les conditions requises pour ces opérations sont décrits dans un document annexe intitulé "Conditions nécessaires au protocole d'écriture et de versionnement distribués pour le World Wide Web" [RFC2291].

Les sections ci-dessous forment une introduction détaillée aux propriétés des ressources (section 4), les collections de ressources (section 5), les opérations de verrouillage (section 6). Ces sections introduisent les abstractions manipulées par les méthodes HTTP spécifiques à WebDav décrites à la section 8, "méthodes HTTP pour l'écriture distribuée".

Dans HTTP/1.1, les paramètres passés aux méthodes étaient exclusivement codés dans les en-têtes HTTP. WebDav, au contraire, code ces informations soit dans le corps d'une entité requête en utilisant le format XML (eXtensible Markup Language) [REC-XML], soit dans un en-tête HTTP. Le choix de l'utilisation de XML pour coder les paramètres d'une méthode a été motivé par la possibilité qu'offre ce langage de pouvoir rajouter des éléments à des structures existantes, apportant ainsi l'extensibilité attendue et par la possibilité qu'offre XML de pouvoir coder l'information en utilisant les jeux de caractères ISO 10646, afin de garantir un support international. Comme règle générique, on retiendra que les paramètres sont codés dans le corps XML des requêtes quand leur longueur est quelconque, ou qu'ils peuvent être vus par un regard humain, ce qui nécessite un codage des caractères en ISO 10646 et qu'ils sont codés à l'intérieur des en-têtes HTTP dans les autres cas. La section 9 décrit les nouveaux en-têtes HTTP utilisés avec les méthodes WebDAV.

Au delà de son utilisation pour le codage des paramètres des méthodes, XML est utilisé dans WebDav pour coder les réponses aux méthodes, apportant ainsi les avantages de l'extensibilité et de l'internationalisation de XML dans les deux cas de figure.

Les éléments XML utilisés dans cette spécification sont définis dans la section 12.

L'extension de l'espaces de noms XML (Annexe 4) est aussi utilisée dans cette spécification afin d'autoriser l'usage de nouveaux éléments XML sans que ceux-là rentrent en collision avec d'autres.

Bien que les codes d'états fournis par HTTP/1.1 soient suffisants pour prendre en charge la plupart des cas d'erreur des méthodes WebDAV, il existe quelques cas d'erreurs qui ne tombent pas nettement dans les catégories existantes. Les nouveaux codes d'états développés pour les méthodes WebDav sont définis dans la section 10. Comme il existe quelques méthodes WebDav qui peuvent agir sur plusieurs ressources, les réponses de type états multiples ont été introduites pour retourner des informations d'états concernant plusieurs ressources. La réponse d'états multiples est décrite

dans la section 11.

WebDAV utilise le mécanisme des propriétés pour renregistrer l'état courant d'une ressource. Par exemple, quand un verrou est posé sur une ressource, une propriété d'information de verrou permet de connaître l'état courant du verrou. La section 13 définit les propriétés utilisées avec les spécifications WebDAV.

La fin de la spécification contient des sections portant sur la compatibilité (section 15), sur le support de l'internationalisation (section 16), et la sécurité (section 17).

# 2. Conventions d'écriture

Ce document décrivant un ensemble d'extensions au protocole HTTP/1.1, l'extension du BNF utilisée ici pour spécifier les éléments du protocole est exactement la même que celle décrite à la section 2.1 de [RFC2068] et s'appuie également sur les règles de production de base fournies à la section 2.2 de [RFC2068].

Les mots clés "**DOIT**", "**NE DOIT PAS**", "**REQUIS**", "**DEVRA**", "**NE DEVRA PAS**", "**DEVRAIT**", **NE DEVRAIT PAS**", "**RECOMMANDÉ**", "**PEUT**" et "**OPTIONNEL**" utilisés dans ce document doivent être interprétés tel que décrit dans le document [RFC2119].

# 3. Terminologie

**URI/URL** : respectivement "Identifiant Uniforme de Ressource" (Uniform Resource Identifier) et "Adresse Uniforme de Ressource" (Uniform Resource Locator). Ces termes (et la distinction entre les deux) sont définis dans le document [RFC2396].

**Collection** : Une collection est une ressource composée d'un ensemble d'URI, celles des membres appelés. Elles identifient les ressources membres et doivent être conformes aux conditions décrites dans la section 5 de cette spécification.

**URI membre** : Toute URI de l'ensemble d'URI constituant une collection.

**URI membre interne** : Une URI membre en relation directe avec l'URI de la collection (la définition de ce que signifie "en relation directe" est écrite à la section 5.2).

**Propriété** : Une paire composée d'un nom et d'une valeur permettant de décrire les caractéristiques de la ressource.

**Propriété vivante** : Une propriété dont la sémantique et la syntaxe sont imposées par le serveur. Par exemple, la valeur de la propriété vivante "`getcontentlength`", en l'occurrence la longueur de l'entité retournée par une requête `GET`, est calculée dynamiquement par le serveur.

**Propriété morte** : Une propriété dont la sémantique et la syntaxe ne sont pas imposées par le serveur. Le serveur ne fait qu'en enregistrer la valeur et il revient au poste client d'en garantir la cohérence syntaxique et sémantique.

**Resource NULL** : Une ressource qui retourne une erreur `404` (message "pas trouvée") en réponse à toute méthode HTTP/1.1 ou DAV sauf dans le cas des méthodes `PUT`, `MKCOL`, `OPTIONS` et `LOCK`.

Une ressource NULL **NE DOIT PAS** apparaître dans la liste des membres de sa collection parente.

# 4. Modèle de données des propriétés des ressources

## 4.1 Modèle des propriétés de ressource

Les propriétés sont une partie des données de la ressource dont elles servent à en décrire l'état. En quelque sorte, les propriétés sont des données qui décrivent des données.

Les propriétés sont utilisées dans les environnements de rédaction distribuée afin de fournir des moyens efficaces de recherche et de gestion des ressources. Par exemple, une propriété qui s'appellerait '`subject`' peut permettre de faire une indexation des ressources par sujet, et une propriété '`author`' offrirait la possibilité de rechercher les documents à partir des noms de leurs auteurs.

Le modèle des propriétés DAV est formé du couple nom/valeur. Le nom d'une propriété permet d'en connaître la syntaxe et la sémantique, et fournit une adresse par laquelle on peut faire référence à cette syntaxe et cette sémantique.

Il y a deux catégories de propriétés : les propriétés "vivantes" et la propriétés "mortes". Une propriété vivante a sa syntaxe et sa sémantique imposées par le serveur. Les propriétés vivantes comprennent les cas où a) la valeur de la propriété est en lecture seule et elle est maintenue par le serveur, et b) la valeur de la propriété est maintenue par le client et le serveur contrôle la syntaxe des valeurs transmises. Toutes les instances d'une propriété vivante **DOIVENT** être conformes à la définition qui a été associée au nom de cette propriété. Une propriété morte a sa syntaxe et sa sémantique imposées par le client; le serveur ne fait bien souvent qu'en enregistrer la valeur mot à mot.

## 4.2 Propositions en cours pour les méta-données

Les propriétés ont toujours joué un rôle très important dans la gestion des grands référentiels de documents, et beaucoup des propositions actuelles intègrent les notions de propriétés, ou sont des discussions relatives au rôle des méta-données sur le Web de manière plus générale. Cela comprend PICS [REC-PICS], PICS-NG, XML, Web Collections, et plusieurs autres propositions concernant la représentation des relations à l'intérieur de HTML. Le travail sur PICS-NG et Web Collections a été poursuivi par le groupe de travail sur RDF (Resource Description Framework) du World Wide Web Consortium. RDF consiste en un modèle de données prévu pour fonctionner en réseau et une représentation XML de ce modèle.

Quelques propositions proviennent des concepts de bibliothèques électroniques. Parmi celles-là se trouve l'ensemble de méta-données Dublin Core [RFC2413] et le Warwick Framework [WF], une structure d'accueil pour différents schémas de méta-données. La littérature contient beaucoup d'exemples de méta-données, par exemple MARC [USMARC], un format de méta-données pour les bibliographies, et le rapport technique sur le format de bibliographie utilisé par le système Dienst [RFC1807]. De plus, les minutes de la première conférence de l'IEEE sur les méta-données fournissent une grande variété d'ensembles de méta-données qui ont été développés pour les besoins spécifiques de communautés particulières.

Les participants à la séance de travail "Metadata II" en 1996 à Warwick, UK [WF], ont pu noter que "de nouveaux ensembles de méta-données se développeront parallèllement à l'évolution de l'infrastructure réseau" et que "différentes communautés proposeront, concevront, et seront responsables de différents types de méta-données". Ces observations peuvent être corroborées en notant que plusieurs ensembles de méta-données spécifiques de communautés particulières existent déjà, et qu'il existe une réelle motivation à développer de nouvelles formes de méta-données au fur et à mesure que le nombre de communautés utilisant des données sous forme numérique augmente, ce qui exige d'avoir un format de méta-données qui permette de les identifier et de les cataloguer.

## 4.3 Propriétés et en-têtes HTTP

Des propriétés exsitent déjà, d'une certaine manière, dans les en-têtes des messages HTTP. Toutefois, dans les environnements de rédaction distribuée, un nombre relativement élevé de propriétés est nécessaire pour décrire l'état d'une ressource. Les initialiser et les retourner par les seuls en-têtes HTTP serait inefficace. Aussi, un mécanisme est nécessaire pour permettre à un "principal" d'identifier l'ensemble des propriétés qui le concernent et qu'il a le droit d'initialiser et de lire.

## 4.4 Valeurs des propriétés

La valeur d'une propriété exprimée en XML **DOIT** être bien formée.

XML a été retenu parce que c'est un format de données flexible, se décrivant lui-même, structuré, qui supporte des définitions de schémas puissants, et supporte plusieurs jeux de caractères. La nature "auto-descriptive" de XML permet à n'importe quelle valeur de propriété d'être étendue par l'ajout de nouveaux éléments. Des clients plus anciens supporteront les extensions parce qu'ils sauront toujours traiter les données spécifiées dans leur format original et ignorer les éléments rajoutés qu'ils ne comprennent pas. Le support des multiples jeux de caractères en XML permet à toute propriété lisible par l'œil humain d'être codée et lue dans un jeu de caractères familier à l'utilisateur. Le support par XML de multiples langages humains, par l'utilisation de l'attribut "xml:lang", permet de gérer les cas où un même jeu de caractères est commun à plusieurs langages humains.

## 4.5 Noms des propriétés

Un nom de propriété est un identifiant universel unique associé à un schéma qui fournit l'information quant à la syntaxe et la sémantique de la propriété.

Une conséquence de l'aspect universel et unique des noms de propriétés est que les données reçues par les applications clientes peuvent dépendre de la régularité du comportement d'une propriété en particulier utilisée dans différentes ressources, que ce soit sur le même serveur ou sur des serveurs différents, et particulièrement si cette propriété est "vivante", le résultat dépendra directement de la fidélité de sa mise en œuvre par rapport à sa définition.

Le mécanisme des espaces de noms de XML, qui repose sur des URI s [RFC2396], est utilisé pour nommer les propriétés parce qu'il permet d'éviter les collisions de noms et fournit plusieurs niveaux de contrôles au niveau de leur gestion.

La propriété `namespace` est plate; c'est à dire qu'elle n'a pas de structure hiérarchique reconnue. C'est à dire que si deux propriétés qui s'écrivent pour l'une `A` et pour l'autre `A/B` existent sur une ressource, cela ne signifie pas qu'il y a une quelconque relation de hiérarchie entre les deux. Une spécification séparée sera éventuellement réalisée sur la question de la hiérarchisation des propriétés.

En conclusion, il n'est pas possible de définir deux fois une même propriété s'appliquant à la même ressource car cela provoquerait une collision dans l'espace de noms des propriétés de cette ressource.

## 4.6 Liens indépendants des médias

Bien que les liens entre ressources soient supportés avec HTML, le Web a besoin d'un support plus général des liens entre les différents types de média possible (ces types de média sont connus comme étant des types MIME ou des types de contenus). WebDav fournit de tels liens. Un lien WebDav est définit par un type spécifique de propriété, formellement défini à la section 12.4 (élément XML `Link`) , qui permet de définir des liens entre des ressources de n'importe quel type de média. La valeur de la propriété est constituée des URI (Uniform Resource Identifiers) des ressources sources et destinations des liens; le nom de la propriété identifie le type de lien.

# 5. Collections de ressources Web

Cette section contient la description d'un nouveau type de ressource Web, le type collection, et discute de ses interactions avec le modèle de l'espace de nom des URL de HTTP. L'objectif d'une ressource de type collection est de modéliser des objets qui sont des regroupements de ressources (par exemple, les répertoires d'un système de fichiers) dans l'espace de noms d'un serveur.

Toutes les ressources compatibles DAV **DOIVENT** supporter le modèle de l'espace de noms des URL de HTTP comme cela est spécifié ci-après.

## 5.1 Modèle de l'espace de noms d'URL HTTP

L'espace de noms d'URL HTTP est un modèle hiérarchique de noms dans lequel la hiérarchie est syntaxiquement représentée par le caractère "/".

Un espace de noms d'URL HTTP est réputé régulier quand il répond à la condition suivante : pour tout URL de la hiérachie HTTP il existe une collection parente contenant cet URL comme membre interne ; seule la racine, ou collection de plus haut niveau de l'espace de noms, est exempté de la règle précédente.

Ni HTTP/1.1 ni WebDav n'imposent que la totalité de l'espace de noms d'URL HTTP soit régulier. Cependant, cette spécification précise que certaines méthodes WebDav ne sont pas autorisées à produire des résultats qui rendrait l'espace de noms irrégulier.

Bien que cela soit implcite dans [RFC2068] et [RFC2396], toute ressource, y compris les ressources de type collection, **PEUVENT** être identifiées par plus d'un URI. Par exemple, une ressource peut être identifiée par plusieurs URL HTTP.

# 5.2 Ressources de type collection

Une collection est une ressource dont l'état est constitué d'au moins une liste d'URI de membres internes et d'un ensemble de propriétés ; une collection peut toutefois avoir des états additionels tel que des corps d'entités retournés suite à l'exécution d'une méthode `GET`. Un URI de membre interne **DOIT** être en relation directe avec un URI de base de la collection. Cela étant, l'URI du membre interne d'une collection est égal à l'URI de la collection auquel on rajoute un segment additionel pour les ressources qui ne sont pas elle-mêmes des collections, ou d'un segment additionel suivi de "`/`" pour les ressources internes à la collection qui sont elles-mêmes des collections, où le segment est défini dans la section 3.3 de la [RFC2396].

Tout URI qui est un membre interne donné **DOIT** appartenir à la collection de manière unique, c'est à dire qu'il est interdit d'avoir plusieurs instances du même URI dans une collection. Les propriétés définies sur des collections ont exactement les mêmes comportements que celles des ressources qui ne sont pas des collections.

Pour toutes resources `A` et `B` compatibles WebDAV, identifiées par les URI `U` et `V`, où `U` est immédiatement relatif à `V`, `B` **DOIT** être une collection ayant U comme URI de membre interne. Ainsi, si les ressources dont les URL http://foo.com/bar/blah et http://foo.com/bar/ sont conformes à WebDav, alors la ressource ayant pour URL http://foo.com/bar/ doit être une collection et doit contenir l'URL http://foo.com/bar/blah comme membre interne.

Dans la hiérarchie des espaces de noms d'URL HTTP, les ressources de type collection **PEUVENT** contenir comme membres internes les URL d'enfants non-conformes à WebDav mais cela n'est pas exigé. Par exemple, si la ressource ayant l'URL http://foo.com/bar/blah n'est pas conforme à WebDAV et que l'URL http://foo.com/bar/ est identifié comme étant une collection alors l'URL http://foo.com/bar/blah peut être ou non un membre interne de la collection http://foo.com/bar/.

Si une ressource conforme à WebDav n'a aucun enfant conforme à WebDav dans l'arborescence de l'espace de noms d'URL HTTP alors WebDav n'exige pas que cette ressource soit une collection.

Il existe une convention établie qui dit que lorsque une collection est référencée par son nom sans la barre oblique de fin, celle-ci est automatiquement rajoutée. À cause de cela, une ressource peut accepter un URI sans "/" de fin pout pointer vers une collection. Dans ce cas, elle **DEVRAIT** retourner dans sa réponse un en-tête `content-location` pointant vers l'URI se finissant par "`/`". Par exemple, si un client invoque une méthode portant sur http://foo.bar/blah (pas de "`/`" à la fin), la ressource http://foo.bar/blah/ (avec un "`/`" de fin) peut répondre comme si la requête lui avait été appliquée, et en renvoyant dans ce cas un en-tête `content-location` contenant http://foo.bar/blah/. En général, les clients **DEVRAIENT** utiliser la forme avec "`/`" pour les noms de collections.

Une ressource **PEUT** être une collection mais n'être pas conforme à WEBDAV. Cela étant, la ressource peut être conforme à toutes les règles définies dans cette spécification quant à la manière dont doit se comporter une collection sans pour autant supporter toutes les méthodes qu'une ressource conforme à WebDav se doit de supporter. Dans ce cas de figure, la ressource peut retourner la propriété `DAV:resourcetype` avec la valeur `DAV:collection` mais **NE DOIT PAS** retourner la valeur "`1`" dans l'en-tête DAV en réponse à la méthode `OPTIONS` .

## 5.3 Création et extraction de ressources collection

Ce document contient la spécification de la méthode `MKCOL` qui sert à créer de nouvelles ressources de type collection, cela en remplacement des méthodes `PUT` ou `POST` d'HTTP/1.1 pour les raisons suivantes :

Dans HTTP/1.1, la définition de la méthode `PUT` est l'enregistrement du corps de la requête à l'endroit spécifié par l'URI de demande. Alors qu'un format de description de collections pourrait tout à fait être conçu à partir de la méthode `PUT`, les conséquences de l'envoi d'une telle description au serveur seraient indésirables. Par exemple, si la description d'une collection, dans laquelle il manquerait des ressources pourtant bien présentes sur le serveur, était mise par `PUT` sur le serveur, cela pourrait être interprété comme une commande de destruction des membres correspondant à l'omission. Cela étendrait la méthode `PUT` jusqu'à effectuer la fonction `DELETE`, ce qui est indésirable car cela changerait la sémantique de la méthode `PUT` et rendrait difficile le contrôle de la fonction `DELETE` par un modèle de gestion des droits d'accès fondé sur les méthodes.

Alors que la méthode `POST` est suffisamment ouverte pour qu'une commande `POST` "créer une collection" puisse être construite, cela est également indésirable parce qu'il serait alors difficile de faire la différence entre le contrôle d'accès pour la création de collection et les autres utilisations de `POST`.

La définition exacte du comportement de `GET` et `PUT` sur les collections est donnée plus loin dans ce document.

## 5.4 Ressources source et ressources de sortie

Dans de nombreux cas de ressources, l'entité retournée par une méthode `GET` correspond exactement à l'état persistant de la ressource, par exemple, un fichier GIF stocké sur un disque. Dans ce cas simple, l'URI auquel la ressource accède est identique à l'URI auquel la source, dans son état persistent, accède. C'est aussi le cas des fichiers HTML sources qui ne sont pas traités par le serveur avant la transmission.

Toutefois, il arrive que le serveur traite les ressources HTML avant leur transmission au client en tant que corps d'entité retourné. Par exemple, une directive de type "`server-side-include`" (SSI) incluse dans un fichier HTML peut donner l'instruction à un serveur de remplacer la directive elle-même par une valeur calculée telle que la date courante. Dans ce cas, le fichier retourné par la méthode `GET` (le fichier HTML d'origine avec le résultat du calcul de la date) est différent de l'état persistent de la ressource (le fichier HTML d'origine contenant la directive SSI). Il est important de remarquer que dans ce cas typique, il n'y a actuellement aucun moyen d'accéder la ressource HTML source qui contient la directive avant traitement.

Parfois, l'entité retournée par la méthode `GET` est le résultat d'un programme d'extraction de données décrites par une ou plusieurs ressources sources (qui peuvent même ne pas avoir d'emplacement à l'intérieur même de l'espace de noms de l'URI). Un seul programme d'extraction de données peut, à lui seul, générer dynamiquement un nombre potentiellement important de ressources de sortie. Un exemple caractéristique peut être celui d'un script CGI qui décrit un programme de type passerelle d'indexation mettant en correspondance une partie de l'espace de nom d'un serveur avec une demande d'indexation, par exemple
http://www.foo.bar.org/finger_gateway/user@host.

Dans le cas où les fonctions de rédaction distribuée ne sont pas supportées, il est acceptable de ne pas établir de correspondance entre les ressources sources et l'espace de nom URI. En faite, cette forme de protection par rapport aux accés à des ressources sources a un réel effet bénéfique sur la sécurité. Toutefois, si l'édition à distance de la ressource source est souhaitée, elle doit avoir un emplacement dans l'espace de noms de l'URI. L'emplacement de la source ne doit pas se trouver au même endroit que celui où la ressource calculée est accessible, puisque, en général, les serveurs sont incapables, dans ce cas, de faire la distinction entre les demandes qui s'adressent aux ressources sources et celles qui concernent les ressources de sortie calculées. Il y a souvent une relation de type n-n entre les ressources sources et les ressources de sortie.

Sur les serveurs conformes à WebDav l'URI de la ressource source peut être enrégistré comme un lien vers la ressource de sortie ayant le type `DAV:source` (se référer à la section 13.10 pour une description de la propriété `link` de la source). Conserver les URI de source sous la forme de liens vers les ressources de sortie revient à faire porter au client la charge de découvrir l'emplacement où se trouve réellement la source sur laquelle se fait la rédaction. Remarquer qu'il n'est pas garanti que la valeur du lien source pointe vers la bonne source. Les liens vers les sources peuvent être rompus ou des erreurs peuvent avoir été commises au moment de leur saisie. Remarquer également que tous les serveurs ne permettront pas aux clients de fixer les valeurs des liens vers les sources. Par exemple un serveur qui génèrerait pour ses fichiers CGI des liens calculés à la volée (par programme) vers les sources n'autoriserait probablement pas un client à régler la valeur du lien source.

# 6. Verrouillage

La possibilité de verrouiller une ressource est un mécanisme permettant de sérialiser les accès à cette ressource. En utilisant un verrou, un client de rédaction fournit une garantie raisonnable qu'un autre principal ne viendra pas modifier une ressource qui serait déjà en cours de modification. En faisant cela, un client est protégé contre le problème de "dernière mise à jour perdue".

Cette spécification permet d'avoir différents types de verrous au moyen de deux paramètres spécifiques aux clients, le nombre de principals impliqués (exclusif vs. partagé) et le type d'accés à leur accorder. Ce document définit le verrouillage pour un seul type d'accès : celui en écriture. Toutefois, la syntaxe est extensible et permet la spécification éventuelle de verrouillage pour d'autres types d'accès.

## 6.1 Verrous exclusifs et partagés

La forme de verrou la plus élémentaire est le verrou exclusif. Il s'agit d'un verrou par lequel le droit d'accès fourni est accordé à un seul principal. La raison de cet arbitrage vient du désir déviter d'avoir à fusionner les résultats.

Toutefois, il y a des cas où le but d'un verrou n'est pas d'en empêcher d'autres d'exercer leur droit d'accès mais plutôt de fournir un mécanisme par lequel les principaux peuvent signaler leur intention d'exercer leur droit d'accès. Les verrous partagés ont été définis précisément pour ce cas de figure. Un verrou partagé permet à plusieurs principals de recevoir un verrou. Dès lors, tout principal ayant un droit d'accès approprié peut obtenir le verrou.

Avec les verrous partagés, il y a deux ensembles d'autorisations pouvant toucher une ressource. Le premier est celui des droits d'accès. Les principals qui sont autorisés peuvent avoir, par exemple, le

droit en écriture dans la ressource. Parmi eux, ceux qui ont obtenu un verrou partagé **DOIVENT** aussi s'autoriser mutuellement : ils constituent alors un sous-ensemble de l'ensemble des droits d'accès.

En considérant tous les principals possible d'Internet, la grande majorité d'entre eux n'auront, dans la pluplart des cas, aucun droit d'accès à une ressource donnée. Parmi le petit nombre qui aura un droit d'accès en écriture, quelques principals voudront s'assurer que leurs éditions seront dégagées de tout risque d'écrasement de données et utiliseront les verrous d'écriture exclusifs. D'autres pourront décider qu'ils ont confiance que leurs collègues n'écraseront pas leur travail (l'ensemble potentiel des collègues étant l'ensemble des principals qui ont les droits en écriture) et utiliseront un verrou partagé, qui permet d'informer les collègues qu'un principal est peut être déjà en train de travailler sur la ressource.

Les extensions WebDav de HTTP n'ont pas besoin de fournir tous les moyens de communication nécessaires aux principals pour coordonner leur activité. Quand on utilise des verrous partagés, les principals ont le droit d'utiliser n'importe quel moyen de communication pour coordoner leurs travaux (par exemple, des notes écrites, des réunions, des post-it collés sur les écrans, des conversations téléphoniques, des courriels, etc.). Le but d'un verrou partagé est d'offrir aux collaborateurs la possibilité de connaître l'identité de celui qui travaille déjà sur la ressource.

Les verrous partagés font partie de cette spécification parce que les expériences passées sur les environnements de rédaction collaborative sur le WEB ont montré que les verrous exclusifs sont trop souvent rigides. Un verrou exclusif est utilisé pour imposer un processus rédactionnel particulier : pose d'un verrou exclusif, lecture de la ressource, modification du contenu, écriture de la ressource, libération du verrou. Mais les verrous ne sont pas toujours correctement libérés, par exemple, quand un programme "se plante", ou quand le propriétaire d'un verrou abandonne son travail en cours de route sans déverrouiller la ressource. Alors que l'utilisation de contrôle de temps limite (timeout) et l'intervention humaine d'un administrateur sont les moyens pour supprimer le verrou indésirable, il peut se trouver qu'aucun de ces deux mécanismes ne soit disponible au moment où on en a justement besoin; le contrôle de dépassement de temps peut être long et l'administrateur indisponible.

# 6.2 Support requis

Il n'est pas exigé que le serveur soit conforme à WebDav pour qu'il sache supporter le verrouillage, quelle qu'en soit la forme. Si le serveur supporte les mécanismes de verrouillage, il peut choisir de supporter n'importe quelle combinaison des formes exclusives et partagées des verrous et pour n'importe quel type d'accès.

La raison de cette souplesse est que les politiques de verrouillage sont intégrées au coeur même des systèmes de gestion et de versionnement des ressources utilisés par différents référentiels de stockage. Ces référentiels ont besoin d'avoir le contrôle sur les types de verrous qui seront disponibles. Par exemple, certains référentiels ne supportent que les verrous d'écriture partagés tandis que d'autres ne supportent que les verrous d'écriture exclusifs et d'autres encore n'en utilisent aucun. Comme chacun de ces systèmes est suffisamment différent pour pouvoir se satisfaire de l'absence de certaines fonctions de verrouillage, cette spécification laisse la question du verrouillage comme étant la seule négotiation possible à l'intérieur de WebDAV.

## 6.3 Marqueurs de verrous

Un jeton de verrou est un type de jeton d'état, représenté comme un URI, qui identifie un verrou particulier. Un jeton de verrou est retourné, après toute opération de verrouillage réussie, dans la propriété `lockdiscovery` du corps de la réponse. Il peut également être retrouvée en lançant une recherche de verrou sur la ressource.

Les URI de marqueurs de verrous **DOIVENT** être uniques au monde pour toutes les resources. Cette contrainte d'unicité permet aux marqueurs de verrous d'être utilisés pour toutes les ressources et les serveurs sans crainte de confusion.

Cette spécification fournit un modèle d'URI de jeton de verrou appelé `opaquelocktoken` conforme à la contrainte d'unicité. Toutefois, les ressources restent libres de retourner n'importe quel modèle d'URI tant qu'il reste conforme à la contrainte d'unicité.

Le fait de pouvoir acccéder aux marqueurs de verrous ne signifie pas pour autant que le principal dispose de droits d'accès privilégiés. Tout le monde peut trouver le jeton de verrou de n'importe quelle autre personne en exécutant simplement une recherche de marqueur. Les verrous **DOIVENT** être appliqués en fonction des mécanismes d'authentification disponibles sur le serveur, mais ne doivent pas reposer sur une politique de secret des valeurs des marqueurs.

## 6.4 Modèle d'URI de jeton de verrou `opaquelocktoken`

Le modèle d'URI `opaquelocktoken` est conçu pour être unique, dans le temps, à travers toutes les ressources. Grâce à cette unicité, un client peut soumettre un jeton de verrou opaque dans un en-tête `If` sur une ressource différente de celle qui l'a retourné initialement.

Toutes les ressources **DOIVENT** reconnaître le modèle `opaquelocktoken` et, au minimum, reconnaître que le jeton de verrou ne fait pas référence à un verrou en suspens de la ressource.

Afin de garantir l'unicité temporelle pour toutes les ressources, le verrou `opaquelocktoken` nécessite l'utilisation d'un mécanisme reposant sur le principe des identifiants universels uniques (Universal Unique Identifier - UUID-), tel qu'il est décrit dans [ISO-11578].

Les générateurs de verrous `Opaquelocktoken` , toutefois, ont le choix de la manière dont ils créent les marqueurs. Ils peuvent soit générer un nouvel UUID pour chaque jeton de verrou qu'ils créent, soit créer un seul UUID puis lui ajouter des caractères d'extension. Si la deuxième méthode est choisie alors le programme de génération des extensions **DOIT** garantir que la même extension ne sera jamais utilisée deux fois sur le même UUID.

La règle de production de l'UUID est la représentation sous forme de chaîne de caractères d'un UUID, tel que défini dans [ISO-11578]. Noter que l'espace blanche (LWS) n'est pas autorisée entre les éléments de cette règle de production.

```
OpaqueLockToken-URI = "opaquelocktoken:" UUID [Extension]

Extension = path
```

"path" est défini dans la section 3.2.1 de la [RFC2068]

# 6.4.1 Génération du champ nœud sans passer par une adresse IEEE 802

Les UUID, tels que définis dans [ISO-11578], contiennent un champ "nœud" qui contient une des adresses IEEE 802 du serveur. Comme il est dit dans la section 17.8, il y a plusieurs risques relatifs à la sécurité inhérents à l'exposition des adresses IEEE 802 des machines. Cette section fournit un mécanisme de remplacement qui permet de générer le champ "nœud" d'un UUID sans utiliser l'adresse IEEE 802. Les serveurs WebDav **PEUVENT** utiliser cet algorithme pour la création du champ nœud au moment de la génération des UUIDs. Le texte de cette section est issu d'un brouillon sur Internet de Paul Leach et Rich Salz, dont les noms sont rappelés ici afin que leurs travaux soient reconnus comme il se doit.

La solution idéale est d'obtenir un nombre aléatoire de 47 bits de qualité cryptographique, et de l'utiliser comme formant les 47 bits de moindre poids de l'ID du nœud, avec le bit de poids fort du premier octet de l'ID du noeud égal à 1. Ce bit est celui réservé au typage envoi individuel/diffusion groupée, qui ne sera jamais établi dans les adresses IEEE 802 obtenues de cartes réseaux ; par conséquent, il ne pourra jamais y avoir de conflit entre les UUID générés par les machines équipées ou non d'une carte réseau.

Si un système n'a pas de primitive pour générer des nombres alétoire de qualité cryptographiqe, il y a la plupart du temps un grand nombre de sources d'aléa disponibles pour en générer. Les types de données sources listées ci-après dépendent bien sûr du système, mais très souvent on trouve :

- le pourcentage de mémoire utilisé
- la taille de la mémoire en octets
- la taille de la mémoire libre en octet
- la taille des fichiers de pagination ou de swap en octets
- l'espace libre, en octets, des fichiers de pagination ou de swap
- la taille totale, en octets, de l'espace d'adressage virtuel d'un utilisateur
- la taille totale, en octets, de l'espace d'adressage d'un utilisateur
- la taille disque, en octets, du pilote de démarrage
- l'espace disque libre, en octets, du pilote de démarrage
- l'heure courante
- la quantité de temps écoulée depuis que le système a été démarré
- la taille individuelle de différents fichiers
- les dates de création, dernier accès, et dernière modification des fichiers dans différents répertoires système

- différents paramètres d'utilisation des ressoures système
- la position courante du curseur de la souris
- la position courante du point d'insertion
- le nombre courant de processus ou de canaux en cours d'exécution
- les identifiants ou les poignées de la fenêtre d'ordinateur et de la fenêtre active
- la valeur du pointeur de pile de l'appelant
- les identificants de processus et de canal d'invite
- différents compteurs d'activité dépendant de l'architecture spécifique du processeur

(Remarquez que c'est précisément le type de ressources aléatoires mentionnées ci-dessus qui sont utilisées pour alimenter les générateurs de nombres alétoires de qualité cryptographique sur des systèmes dont le matériel n'est pas équipé de fonctions spécifiques pour les générer.)

De plus, des données telles que le nom de l'ordinateur et le nom du système d'exploiation, bien que ne pouvant être qualifiées d'alétoires à proprement parler, pourront aider à faire la diffférence avec les nombres aléatoires produits par d'autres systèmes.

L'algorithme exact de génération d'un identifiant de nœud utilisant ces données est spécifique du système, parce que à la fois les données disponibles et les fonctions pour les obtenir sont souvent très spécifiques du système. Toutefois, en supposant que l'on pourrait concaténer toutes les valeurs des sources d'aléas dans une mpémoire TAMPON, et qu'une fonction de hachage cryptographique telle que MD5 soit disponible, alors tout mot de 6 octets du hachage MD5 de la mémoire tampon, ayant le bit Diffusion groupée établi (le bit de poids fort du premier octet) pourra être un identifiant de noeud d'un aléa acceptable.

D'autres fonctions de hachage, telle que par exemple SHA-1, peuvent aussi être utilisées. La seule obligation est que le résultat soit convenablement aléatoire, c'est à dire que d'un ensemble d'entrées uniformément distribué résulte un ensemble de sorties elles-mêmes uniformément distribuées, et que la modification d'un seul bit en entrée provoque le changement de la moitié des bits de sortie.

# 6.5 Exploration des possibilités de verrouillage

Puisque le support des verrous au niveau du serveur est optionnel, un client essayant de bloquer une ressource sur un serveur peut soit essayer le verrouillage et espérer que cela marche, soit exécuter une sorte d'exploration pour savoir quelles scapacités du serveur prennent en charge le verrouillage. Cela est connu sous le nom d'exploration des possibilités de verrouillage. Cela est différent de la recherche des types de contrôles d'accès supportés puisque il peut y avoir des types de contrôles d'accès sans verrou correspondant. Un client peut déterminer quels sont les types de verrous supportés par le serveur en récupérant cette information par la propriété `supportedlock`.

Toute ressource conforme à DAV qui supporte la méthode LOCK **DOIT** supporter la propriété

supportedlock.

## 6.6 Exploration des verrous actifs

Si un principal verrouille la ressource à laquelle un autre principal espère accéder, il est pratique pour le deuxième d'être capable de trouver qui est le premier. Pour cela, il existe la propriété lockdiscovery. Elle liste tous les verrous en cours, décrit leur type, et, quand ils sont disponibles, fournit les marqueurs de verrous qui s'y appliquent.

Toute ressource conforme à DAV qui supporte la méthode LOCK **DOIT** supporter la propriété lockdiscovery.

## 6.7 Considérations d'utilisation

Bien que les mécanismes de verrouillage spécifiés dans ce document fournissent une certaine garantie contre le problème de la dernière mise à jour perdue pour cause d'écrasement, ils ne peuvent cependant pas garantir à 100% que les mises à jour ne seront jamais perdues. Imaginez le scénario suivant :

Deux clients A et B souhaitent éditer la même ressource "index.html". Le client A utilise le protocole HTTP plutôt que WebDav , et donc, ne sait pas gérer les verrous. Le client A ne verrouille pas le document, exécute un GET puis commence l'édition. Le client B fait un LOCK , exécute un GET et commence l'édition. Le client B termine l'édition, exécute un PUT , puis un UNLOCK . Le client A exécute un PUT , écrase le fichier et fait perdre à B ses mises à jour.

Il y a plusieurs raisons pour lesquelles le protocole WebDav lui-même ne peut pas anticiper cette situation. La première est qu'il ne peut pas obliger tous les clients à utiliser le verrouillage parce qu'il doit rester compatible avec les clients HTTP qui, eux, ne comprennent pas le verrouillage. Deuxièmement, il ne peut pas exiger des serveurs qu'ils supportent le verrouillage à cause de la variété des mises en œuvre de référentiels, les uns s'appuyant plutôt sur des mécanismes de réservation et de fusion que sur des verrous. Finallement, parce que le protocole est de type "sans gestion d'état" (stateless), il ne peut obliger le client à exécuter une séquence d'opérations prédéfinie telle que LOCK / GET / PUT / UNLOCK.

Les serveurs WebDav qui supportent le verrouillage réduisent le risque d'avoir des clients écrasant mutuellement et par accident leurs modifications respectives en exigeant d'eux qu'ils bloquent les ressources avant de les modifier. De tels serveurs protègeront effectivement les clients HTTP 1.0 et HTTP 1.1 des modifications indésirables des ressources.

Les clients WebDav peuvent être de bons citoyens utilisant la séquence d'opérations lock / retrieve / write /unlock (au moins par défaut) chaque fois qu'ils interagiront avec un serveur WebDav qui supportera le verouillage.

Les clients HTTP 1.1 peuvent être de bons citoyens, c'est à dire en évitant d'écraser les changements faits par d'autres clients, en utilisant des balises d'entités dans les en-têtes If-Match dans toute demande qui modifierait des ressources.

Les gestionnaires d'informations peuvent essayer d'empêcher les écrasements en mettant en œuvre

des procédures du côté client exigeant le verrouillage avant de modifier des ressources WebDav .

# 7. Verrou d'écriture

Cette section décrit la sémantique propre au verrou d'écriture. Le verrou d'écriture est une instance spécifique du type verrou ; il est le seul type de verrou décrit dans cette sépcification.

## 7.1 Méthodes restreintes par les verrous d'écriture

Un verrou d'écriture **DOIT** contrôler qu'un principal n'ayant pas le verrou ne peut exécuter avec succès aucune des méthodes `PUT, POST, PROPPATCH, LOCK, UNLOCK, MOVE, DELETE` , ou `MKCOL` sur la ressource verrouillée. Toutes les autres méthodes courantes, en particulier `GET` , fonctionnent indépendamment de ce verrou.

Remarquez, toutefois, que lorsque des nouvelles méthodes seront créées, il sera nécessaire de spécifier comment elles devront se comporter vis à vis du verrou d'écriture.

## 7.2 Verrous d'écriture et marqueurs de verrous

Une demande réussie pour obtenir un verrou d'écriture **DOIT** se traduire par la génération d'un jeton de verrou unique associé au principal. Aussi, si cinq principaux ont un verrou d'écriture partagé sur la même ressource il y aura cinq marqueurs de verrous, un par principal.

## 7.3 Verrous d'écriture et propriétés

Alors que ceux qui n'ont pas posé de verrou d'écriture ne peuvent pas modifier les propriétés d'une ressource, il est cependant toujours possible que les valeurs des propriétés vivantes soient changées, même quand elles sont verrouillées, à cause des exigences de leurs modèles.

Seules les propriétés mortes, et les propriétés vivantes définies comme respectant les verrous, ont la garantie de ne pas changer quand un verrou d'écriture est posé.

## 7.4 Verrous d'écriture et ressources nulles

Il est possible de positionner un verrou d'écriture sur une ressource nulle afin d'en bloquer le nom.

Une ressource nulle bloquée en écriture, connue sous le nom de ressource nulle verrouillée, **DOIT** retourner une erreur type `404` (Pas trouvée) ou `405` (méthode interdite) à toute méthode HTTP/1.1 ou `DAV` à l'exception des méthodes `PUT, MKCOL, OPTIONS, PROPFIND, LOCK` , et `UNLOCK` . Une ressource nulle verrouillée **DOIT** apparaître comme un membre de la collection parente dont elle fait partie. De plus, la ressource nulle verrouillée **DOIT** être équipée de toutes les propriétés obligatoires de `DAV` . La plupart de ces propriétés, comme par exemple toutes les propriétés de type `get*` , n'auront aucune valeur puisqu'une ressource nulle verrouillée ne supporte pas la méthode `GET`. Les ressources nulles verrouillées **DOIVENT** être équipées des propriétés

lockdiscovery et supportedlock .

Jusqu'au moment où l'une des méthodes PUT ou MKCOL est exécutée avec succès sur la ressource nulle verrouillée, elle **DOIT** rester dans son état de ressource nulle verrouillée. Toutefois, dès qu'une des méthodes PUT ou MKCOL est exécutée avec succès, alors la ressource nulle verrouillée cesse d'être dans l'état nul-verrouillé.

Si la ressource est déverrouillée, quelle qu'en soit la raison, sans qu'un PUT,MKCOL , ou toute méthode similaire ait été exécutée avec succès, la ressource **DOIT** alors retourner à son état nul.

# 7.5 Verrous d'écriture et collections

Un verrou d'ériture sur une collection, qu'il ait été créé par une requête de verrouillage de type "Depth: 0 " ou "Depth: infinity ", empêche le rajout ou le retrait d'URI membres de la collection par d'autres principaux que les propritaires des verrous. En conséquence, quand un principal réalise une requête PUT ou POST pour créer une nouvelle ressource sous un URI qui se trouve être un membre interne d'une collection verrouillée en écriture et pour maintenir la cohérence de l'espace de noms HTTP , ou qu'il souhaite réaliser un DELETE pour retirer une ressource dont l'URI est un URI d'un membre interne d'une collection protégée en écriture, alors cette requête **DOIT** échouer si le principal n'est pas propriétaire d'un verrou d'écriture au niveau de la collection.

Toutefois, si une demande de verrou d'écriture est faite sur une collection contenant des URI membres qui s'appliquent à des ressources en cours de verrouillage et d'une manière qui soit en conflit avec le verrou d'écriture, alors la demande **DOIT** échouer et retourner le code d'état 423 (Verrouillé).

Si le propriétaire d'un verrou rajoute l'URI d'une ressource en tant que membre interne d'une collection verrouillée alors cette nouvelle ressource **DOIT** être automatiquement rajoutée au verrou. C'est le seul mécanisme qui permet de rajouter une ressource à un verrou d'écriture. Aussi, par exemple, si la collection /a/b/ est verrouillée en écriture et si la ressource /c est déplacée pour devenir /a/b/c alors la ressource /a/b/c sera rajoutée au verrou d'écriture.

# 7.6 Verrous d'écriture et en-tête de requête If

Si il n'est pas obligatoire d'avoir un agent utilisateur pour connaître l'existence d'un verrou quand on demande l'exécution d'une opération sur une ressource verrouillée, le scénario suivant peut se produire. Un programme A, exécuté par un utilisateur A, pose un verrou d'écriture sur une ressource. Un programme B, également exécuté par l'utilisateur A, n'a pas la connaissance du verrou posé par le programme A, exécute un PUT sur la ressource verrouillée. Dans ce scénario, le PUT est réalisé avec succès parce que les verrous sont associés à un même principal, pas un programme. Et donc le programme B, parce qu'il est activé par le même certificat de principal que A, est autorisé à faire le PUT . Toutefois, si le programme B avait eu connaissance du verrou, il n'aurait pas écrasé la ressource, préférant à la place présenter une boîte de dialogue annonçant le conflit à l'utilisateur. À cause de ce scénario, on a besoin d'un mécanisme pour empêcher que des programmes différents ignorent accidentellement les verrous posés par d'autres programmes ayant les mêmes privilèges.

Afin d'empêcher ces collisions, un jeton de verrou **DOIT** être posé par un principal autorisé dans

l'en-tête `If` de toute ressource verrouillée pouvant réagir à des méthodes ou sinon la méthode **DOIT** échouer. Par exemple, si une ressource doit être déplacée et si la source et la destination sont toutes les deux verrouillées, alors deux marqueurs de verrous doivent être soumis, l'un pour la source et l'autre pour la destination.

## 7.6.1 Exemple - verrou d'écriture

>>Requête

```
COPY /~fielding/index.html HTTP/1.1 Host: www.ics.uci.edu
Destination: http://www.ics.uci.edu/users/f/fielding/index.html
If: <http://www.ics.uci.edu/users/f/fielding/index.html>
(<opaquelocktoken:f81d4fae-7dec-11d0-a765-00a0c91e6bf6>)

>>Réponse

HTTP/1.1 204 No Content
```

Dans cet exemple, même si la source et la destination sont toutes les deux verrouillées, un seul jeton de verrou doit être soumis, celui pour le verrou sur la destination. Cela parce que la source n'est pas modifiée par l'ordre `COPY` , et par conséquent, n'est pas concernée par le verrou d'écriture. Dans cet exemple, l'authentification de l'agent utilisateur est déjà intervenue via un mécanisme en dehors du périmètre concerné par le protocole HTTP , c'est à dire dans la couche de transport sous-jacente.

## 7.7 Verrous d'écritre et méthode `COPY/MOVE`

L'exécution de la méthode `COPY` **NE DOIT PAS** dupliquer les verrous d'écriture actifs sur la source. Toutefois, comme il a été noté précédemment, si l'ordre `COPY` copie la ressource dans une collection qui, elle, est verrouillée avec l'option "`Depth: infinity` ", alors la ressource sera ajoutée au verrou.

Une requête `MOVE` appliquée avec succès sur une ressource ayant un verrou d'écriture **NE DOIT PAS** déplacer le verrou d'écriture avec la ressource. Toutefois, la ressource est sujette à être rajoutée à un verrou déjà existant au niveau de la destination, comme cela est spécifié dans la section 7. Par exemple, si le `MOVE` fait que la ressource devient un enfant d'une collection qui est verrouillée avec l'option "Depth: infinity", alors la ressource sera ajoutée au verrou déjà existant au niveau de la collection. De plus, si une ressource verrouillée avec l'option "`Depth: infinity` " est déplacée vers une destination qui se trouve a l'intérieur de la portée de ce même verrou (par exemple, à l'intérieur de l'arbre d'espace de noms couvert par le verrou) la ressource déplacée sera à nouveau rajoutée au verrou. Dans ces deux exemples, et comme il est spécifié dans la section 7.6, un en-tête `If` contenant un jeton de verrou doit être soumis tant pour la source que pour la destination.

## 7.8 Rafraîchissement des verrous d'écriture

Un client **NE DOIT PAS** soumettre deux fois de suite une même demande de verrouillage en écriture. Remarquez qu'un client sait toujours qu'il est en train de soumettre la même demande de verrouillage pour la deuxième fois puisqu'il doit inclure le jeton de verrou dans l'en-tête `If` afin de

pouvoir soumettre sa requête sur une ressource qui est déjà verrouillée.

Toutefois, un client peut soumettre une méthode `LOCK` avec un en-tête `If` mais sans corps. Cette forme de `LOCK` **DOIT** être uniquement utilisée pour "rafraichir" un verrou. Cela signifie, au minimum, que tout temporisateur associé au verrou **DOIT** être réinitialisé.

Un serveur peut très bien retourner un en-tête Timeout avec un rafraîchissement de verrou différent de l'en-tête Timeout retourné quand le verrou fut initialement créé. De plus, des clients peuvent soumettre des en-têtes Timeout de valeur arbitraire dans leurs requêtes de rafraîchissement de verrou. Les serveurs, comme toujours, peuvent ignorer les en-têtes Timeout soumis par leurs clients.

Si une erreur est reçue en réponse à une requêtre de rafraîchissement de `LOCK` le client **DEVRAIT** supposer que le verrou n'a pas été rafraichi.

# 8. Méthodes HTTP pour la rédaction distribuée

Les nouvelles méthodes HTTP décrites ci-après utilisent XML comme format de demande et de réponse. Tous les clients et les ressources conformes à DAV **DOIVENT** utiliser des analyseurs XML conformes à la norme [REC-XML]. Tout le XML utilisé que ce soit dans les demandes ou dans les réponses **DOIT** être, au minimum, bien formé. Si un serveur reçoit dans une requête des données XML mal formées il **DOIT** rejeter la totalité de la requête avec un code `400` (mauvaise requête). Si un client reçoit dans une réponse des données XML mal formées, il **NE DOIT PAS** supposer quoi que ce soit concernant le résultat de la méthode exécutée et **DEVRAIT** considérer que le serveur ne fonctionne pas bien.

## 8.1 La méthode PROPFIND

La méthode `PROPFIND` permet de récupérer les propriétés de la ressource identifiée par l'URI de demande, si la ressource n'a aucun membre interne, ou celles de la ressource identifiée par l'URI de demande et éventuellement ses ressources membres internes si la ressource est une collection qui a des URI de membres internes. Toutes les ressources conformes à DAV **DOIVENT** supporter la méthode `PROPFIND` et l'élément XML `propfind` (section 12.14) ainsi que tous les éléments XML définis pour être utilisés avec cet élément.

Un client peut soumettre un en-tête `Depth` avec une des valeurs "`0` ", "`1` " ou "`infinity` " avec la méthode `PROPFIND` appliquée à une ressource de type collection contenant des URI de membres internes. Les serveurs conformes à DAV **DOIVENT** supporter les comportements "`0` ", "`1` " et "`infinity` ". Par défaut, une méthode `PROPFIND` sans en-tête `Depth` **DOIT** agir comme si en en-tête "`Depth: infinity` " était inclus.

Un client peut soumettre un élément XML `propfind` dans le corps de la méthode de demande `qui décr`it l'information recherchée. Il est possible de demander des valeurs de propriétés particulières, toutes les valeurs de propriétés, ou une liste des noms des propriétés de la ressource. Un client peut choisir de ne pas soumettre un corps de demande. Un corps de demande `PROPFIND` vide **DOIT** être traité comme une demande des noms et des valeurs de toutes les propriétés.

Tous les serveurs **DOIVENT** être capables de retourner une réponse dont le contenu soit de type

`text/xml` ou `application/xml` et qui contient un élément XML `multistatus` qui décrit les résultats des tentatives de restitution des différentes propriétés.

Si un problème survient dans la lecture d'une propriété, alors une erreur spécifique **DOIT** être incluse dans la réponse. Une requête qui porte sur l'obtention de la valeur d'une propriété qui n'existe pas est un cas d'erreur qui **DOIT** être relevé, si la réponse utilise un élément XML `multistatus` , avec un élément XML de réponse contenant le code d'état `404` (Pas Trouvé).

En conséquence, l'élément XML `multistatus` d'une ressource de type collection ayant des URI membres **DOIT** avoir, pour toute profondeur spécifiée, un élément XML `response` pour chacun des URI membres de la collection. Chaque élément XML `response` **DOIT** contenir l'élément XML `href` qui donne l'URI de la ressource à laquelle se rapportent les propriétés de l'élément XML `prop`. Les résultats de la méthode `PROPFIND` appliquée à une ressource de type collection ayant des URI de membres internes sont retournés sous la forme d'une liste plate dont l'ordre des entrées n'est pas significatif.

Dans le cas de `allprop` et `propname`, si un principal n'a pas le droit de savoir si une propriété particulière existe, la propriété devrait alors être exclue en silence de la réponse.

Les résultats de cette méthode **NE DEVRAIENT PAS** être mis en antémémoire.

# 8.1.1 Exemple - Obtenir des propriétés nommées

>>Requête

```
PROPFIND /file HTTP/1.1 Host: www.foo.bar Content-type: text/xml;
charset="utf-8" Content-Length: xxxx

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

>>Réponse

```
HTTP/1.1 207 Multi-Status Content-Type: text/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.foo.bar/file>           <D:propstat>
               <D:prop xmlns:R="http://www.foo.bar/boxschema/">
                    <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 Forbidden</D:status>
                <D:responsedescription> The user does not have
access to the DingALing property.
</D:responsedescription>           </D:propstat>
</D:response>      <D:responsedescription> There has been an
access violation error.</D:responsedescription>
</D:multistatus>
```

Dans cet exemple, la méthode `PROPFIND` est exécutée sur une ressource qui n'est pas une collection, à savoir le fichier http://www.foo.bar/file. L'élément XML `propfind` précise le nom de quatre propriétés dont on cherche à obtenir les valeurs. Dans ce cas, seulement deux des valeurs sont retournées, puisque le principal qui émet cette requête n'a pas les droits d'accès suffisants pour voir les troisième et quatrième propriétés.

## 8.1.2 Exemple - Utilisation de `allprop` pour obtenir toutes les propriétés

>>Requête

```
PROPFIND /container/ HTTP/1.1 Host: www.foo.bar Depth: 1 Content-
Type: text/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: text/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.foo.bar/container/>           <D:propstat>
                <D:prop xmlns:R="http://www.foo.bar/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:resourcetype><D:collection/></D:resourcetype>
                     <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>http://www.foo.bar/container/front.html>
<D:propstat>                  <D:prop
xmlns:R="http://www.foo.bar/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 resource
                    </D:displayname>
<D:getcontentlength>                             4525
                    </D:getcontentlength>
<D:getcontenttype>                              text/html
                    </D:getcontenttype>
<D:getetag>                             zzyzx
</D:getetag>                       <D:getlastmodified>
                         Monday, 12-Jan-98 09:25:56 GMT
                    </D:getlastmodified>
<D:resourcetype/>                       <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, la méthode `PROPFIND` a été invoquée sur la ressource http://www.foo.bar/container/ avec un en-tête `Depth` de `1` , signifiant que la requête s'applique à la ressource et ses enfants, et un élément XML `propfind` qui contient l'élément XML `allprop` , ce qui signifie que la requête devrait retourner le nom et la valeur de toutes les propriétés définies sur chaque ressource.

La ressource http://www.foo.bar/container/ possède les six propriétés suivantes :

1. http://www.foo.bar/boxschema/bigbox,
2. http://www.foo.bar/boxschema/author,
3. `DAV:creationdate` ,

4. DAV:displayname,
5. DAV:resourcetype ,
6. DAV:supportedlock .

Les quatre dernières sont spécifiques de WebDAV, et sont définies dans la section 13. Comme la méthode GET n'est pas supportée par cette ressource, les propriétés get* (comme par exemple getcontentlength ) ne sont pas définies sur cette ressource. Les propriétés spécifiques de DAV attestent que "container " fut créé le 1er décembre 1997, à 5:42:21 de l'après midi, dans un fuseau horaire à 8 heures ouest de GMT (creationdate), a le nom "exemple collection " (displayname ), un type de ressource collection (resourcetype ), et supporte les verrous d'écriture exclusifs et partagés (supportedlock ).

La ressource http://www.foo.bar/container/front.html possède les neuf propriétés suivantes :

1. http://www.foo.bar/boxschema/bigbox (un autre cas d'instance du type de propriété "bigbox "),
2. DAV:creationdate,
3. DAV:displayname,
4. DAV:getcontentlength,
5. DAV:getcontenttype,
6. DAV:getetag,
7. DAV:getlastmodified,
8. DAV:resourcetype ,
9. DAV:supportedlock .

Les propriétés spécifiques de DAV attestent que "front.html " a été créé le 1er décembre 1997, à 6:27:21 de l'après-midi, dans un fuseau horaire de 8 heures à l'ouest de GMT (creationdate ), a le nom de "exemple HTML resource " (displayname ), a un contenu de 4525 octets de long (getcontentlength ), a le type MIME "text/html " (getcontenttype ), a une étiquette d'entité "zzyzx " (getetag ), a été modifié en dernier le lundi 12 janvier 1998 à 09:25:56 GMT (getlastmodified ), a un type de ressource vide, signifiant qu'il ne s'agit pas d'une collection (resourcetype ), et accepte les deux types de verrous d'écriture : exclusif et partagé (supportedlock ).

## 8.1.3 Exemple - Utilisation de propname pour obtenir les noms de toutes les propriétés

>>Requête

```
PROPFIND /container/ HTTP/1.1 Host: www.foo.bar Content-Type:
text/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: text/xml; charset="utf-8"
Content-Length: xxxx

   <?xml version="1.0" encoding="utf-8" ?>    <multistatus
xmlns="DAV:">      <response>
<href>http://www.foo.bar/container/>           <propstat>
               <prop xmlns:R="http://www.foo.bar/boxschema/">
                    <R:bigbox/>                      <R:author/>
                    <creationdate/>
<displayname/>                    <resourcetype/>
                    <supportedlock/>                 </prop>
               <status>HTTP/1.1 200 OK</status>
</propstat>      </response>      <response>
<href>http://www.foo.bar/container/front.html>
<propstat>                 <prop
xmlns:R="http://www.foo.bar/boxschema/">
<R:bigbox/>                       <creationdate/>
                    <displayname/>
<getcontentlength/>                        <getcontenttype/>
                    <getetag/>
<getlastmodified/>                      <resourcetype/>
                    <supportedlock/>                 </prop>
               <status>HTTP/1.1 200 OK</status>
</propstat>      </response>    </multistatus>
```

Dans cet exemple, `PROPFIND` est invoqué sur la ressource collection http://www.foo.bar/container/, avec un élément XML `propfind` contenant l'élément XML `propname` , signifiant que les noms de toutes les propriétés devraient être retournés. Comme aucun en-tête `Depth` n'est présent, il suppose la valeur par défaut "`infinity` " , ce qui signifie que tous les noms des propriétés de la collection mais également de tous ses descendants devraient être retournés.

Ainsi, on trouve que dans l'exemple précédent, la ressource http://www.foo.bar/container/ possède six propriétés :

1. http://www.foo.bar/boxschema/bigbox,
2. http://www.foo.bar/boxschema/author,
3. `DAV:creationdate,`
4. `DAV:displayname,`
5. `DAV:resourcetype` ,

6. DAV:supportedlock .

La ressource http://www.foo.bar/container/index.html, membre de la collection "enveloppe", possède neuf propriétés,

1. http://www.foo.bar/boxschema/bigbox,
2. DAV:creationdate,
3. DAV:displayname,
4. DAV:getcontentlength,
5. DAV:getcontenttype,
6. DAV:getetag,
7. DAV:getlastmodified,
8. DAV:resourcetype ,
9. DAV:supportedlock .

Cet exemple montre également l'utilisation de la portée des espaces de noms de XML, et de l'espace de noms par défaut. Puisque l'attribut "xmlns" n'a pas de lettre de raccourci (préfixe) explicite, l'espace de nom s'applique par défaut à tous les éléments englobés. Par conséquent, tous les éléments qui ne font pas explicitement état de l'espace de noms auquel ils appartiennent sont membres du schéma de l'espace de noms "DAV: ".

# 8.2 Méthode PROPPATCH

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

Toutes les ressources conformes à DAV **DOIVENT** supporter la méthode PROPPATCH et **DOIVENT** traiter les instructions qui sont spécifiées en utilisant les éléments XML propertyupdate, set, et remove du schéma de données de DAV. L'exécution des consignes dans cette méthode est, bien sûr, soumise aux contraintes du contrôle d'accès. Les ressources conformes à DAV **DEVRAIENT** supporter l'iétablissement de propriétés mortes arbitraires.

Le corps de message de demande d'une méthode PROPPATCH **DOIT** contenir l'élément XML propertyupdate . Le traitement des instructions **DOIT** se faire dans l'ordre des instructions reçues (c'est à dire de haut en bas). Les instructions **DOIVENT** être soit toutes exécutées soit ne pas être exécutées du tout. Ainsi, si une erreur survient pendant le traitement alors toutes les instructions déjà exécutées **DOIVENT** être annulées et un code d'erreur approprié doit être retourné. Le traitement détaillé des instructions peut être trouvé dans la définition des instructions set et remove dans la section 12.13.

## 8.2.1 Codes d'états à utiliser avec des méthodes à états multiples 207

Les exemples suivants sont des codes de réponse qu'on s'attend à recevoir dans les réponses aux méthodes de type 207 (états multiples). Noter cependant que tant que cela n'est pas explicitement prohibé, tout code de réponse des séries 2/3/4/5xx peut être utilisé dans une réponse de type 207 (états multiples).

200 (OK) - La commande a réussi.Comme il peut y avoir une série de commandes set et remove dans un corps de requête, un code de retour 201 (Créé) semble inapproprié.

403 (Interdit) - Le client, pour une raison que le serveur choisit de ne pas spécifier, ne peut pas modifier l'une des propriétés.

409 (Conflit) - Le client a transmis une valeur dont la sémantique est inappropriée pour la propriété visée. Cela inclut de tenter d'établir des propriétés de lecture seule.

423 (Verrouillé) - La ressource spécifiée est verrouillée et le client, soit n'est pas le propriétaire du verrou, soit le type de verrou requiert qu'un jeton de verrou soit soumis et le client ne l'a pas fait.

507 (Mémoire insuffisante) - Le serveur n'a pas assez d'espace pour enregistrer la propriété.

## 8.2.2 Exemple - PROPPATCH

>>Requête

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

   <?xml version="1.0" encoding="utf-8" ?>    <D:propertyupdate
xmlns:D="DAV:"    xmlns:Z="http://www.w3.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: text/xml; charset="utf-8"
Content-Length: xxxx

   <?xml version="1.0" encoding="utf-8" ?>    <D:multistatus
xmlns:D="DAV:"    xmlns:Z="http://www.w3.com/standards/z39.50">
     <D:response>           <D:href>http://www.foo.com/bar.html>
          <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>
Copyright Owner can not be deleted or
altered.</D:responsedescription>      </D:response>
</D:multistatus>
```

Dans cet exemple, le client demande au serveur d'établir la valeur de la propriété http://www.w3.com/standards/z39.50/Authors, et de retirer la propriété http://www.w3.com/standards/z39.50/Copyright-Owner. Comme la propriété Copyright-Owner n'a pas pu être retirée, aucune modification de propriété n'est faite. Le code d'état `424` (Echec de dépendance) pour la propriété `Authors` permet de savoir que cette action aurait été réussie si elle n'en n'avait pas été empêchée par le conflit sur la suppression de propriété Copyright-Owner .

## 8.3 Méthode `MKCOL`

La méthode `MKCOL` est utilisée pour créer une nouvelle collection. Toutes les ressources conformes à DAV **DOIVENT** supporter la méthode `MKCOL` .

### 8.3.1 Demande

`MKCOL` crée une nouvelle ressource de type collection à l'emplacement spécifié par l'URI de demande. Si la ressource identifiée par l'URI de demande est non nulle, alors le `MKCOL` **DOIT** échouer. Pendant le traitement `MKCOL`, un serveur **DOIT** faire en sorte que l'URI de demande devienne un membre de sa collection parent, sauf si l'URI de demande est lui-même `"/"`. Si un tel ancêtre n'existe pas, la méthode `MKCOL` **DOIT** échouer. Quand l'opération `MKCOL` crée une nouvelle ressource de collection, tous les ancêtres **DOIVENT** préalabalement exister, sinon, la méthode **DOIT** échouer et retourner le code d'état 409 (Conflit). Par exemple, si une demande de création de la collection `/a/b/c/d/` est faite, et que ni `/a/b/` ni `/a/b/c/` n'existent, alors la demande **DOIT** échouer.

Quand `MKCOL` est invoquée sans corps de requête, la collection nouvellement créée **DEVRAIT** n'avoir aucun membre.

Un message de demande `MKCOL` peut contenir un corps. Le comportement d'une requête `MKCOL` ayant un corps est limité à la création de collections, de membres d'une collection, de corps de membres et de propriétés sur les collections et les membres. Si le serveur reçoit un type d'entité de demande MKCOL qu'il ne supporte pas ou qu'il ne comprend pas il **DOIT** retourner le code d'état 415 (Type de support non pris en charge). Le comportement exact de `MKCOL` pour les divers types de support de demande est indéfini dans ce document, et sera spécifié dans d'autres documents.

### 8.3.2 Codes d'états

Les réponses aux demandes `MKCOL` **NE DOIVENT PAS** être mises en antémémoire car MKCOL a une sémantique non idempotente.

`201` (Créée) - La collection ou la ressource structurée a été correctement et complètement créée.

403 (Interdit) - Ce code indique qu'au moins une des deux conditions est vérifiée :

1. le serveur n'autorise pas la création de collections à l'emplacement indiqué dans son espace de noms,

2. la collection parente de l'URI de demande existe mais ne peut recevoir de membres.

405 (Méthode non autorisée) - MKCOL peut seulement être exécuté sur une ressource détruite ou inexistante.

409 (Conflit) - Une collection ne peut pas être fabriquée à l'URI de demande spécifiée tant que au moins une collection intermédiaire n'aura pas été créée.

415 (Type de support non accepté)- Le serveur ne prend pas en charge le type de demande du corps.

507 (Mémoire insuffisante) - Il n'y a pas assez d'espace pour que la ressource puisse enregistrer son état après exécution de la méthode.

## 8.3.3 Exemple - MKCOL

Cet exemple crée une collection nomée /webdisc/xfiles/ sur le serveur www.server.org.

>>Requête

```
MKCOL /webdisc/xfiles/ HTTP/1.1 Host: www.server.org
```

>>Réponse

```
HTTP/1.1 201 Created
```

## 8.4 Méthodes GET et HEAD appliquées à des collections

La sémantique de la méthode GET est inchangée quand elle est appliquée à une collection, puisque la méthode GET est définie ainsi, "extrait toute information (sous la forme d'une entité) identifiée par l'URI de demande" [RFC2068]. Quand la méthode GET est appliquée à une collection, elle peut retourner le contenu d'une ressource "index.html", ou encore une représentation humainement lisible du contenu de la collection, ou d'autres résultats encore. Par conséquent, il est tout à fait possible que le résultat d'un GET appliqué à une collection n'aura aucun rapport avec les membres appartenant à cette collection.

De la même manière, comme la définition de HEAD est un GET sans corps de message de réponse, la sémantique de HEAD est inchangée quand appliquée à des ressources de collection.

## 8.5 Méthode POST appliquée à des collections

Puisque par définition la fonction réellement réalisée par POST est déterminée par le serveur et

dépend souvent de la ressource particulière à laquelle elle est appliquée, le comportement de `POST` appliqué à des collections ne peut pas être significativement modifié parce qu'il est largement indéfini. En conséquence la sémantique de l`POST` est inchangée quand elle s'applique à une collection.

## 8.6 Méthode `DELETE`

## 8.6.1 Méthode `DELETE` appliquée à des ressources autres que des collections

Si la méthode `DELETE` est appliquée à une ressource qui n'est pas une collection et dont l'URI est un membre interne d'une ou plusieurs collections, alors un serveur **DOIT**, pendant l'exécution du `DELETE`, retirer des collections qui la contiennent comme membre tous les URI de la ressource identifiée par l'URI de demande.

## 8.6.2 Méthode  DELETE appliquée à des collections

La méthode DELETE appliquée à une collection **DOIT** agir comme si l'en-tête "Depth: infinity" lui était appliqué. Un client **NE DOIT PAS** soumettre un en-tête Depth avec un DELETE sur une collection avec une autre valeur que infinity.

DELETE donne l'instruction de suppression de la collection spécifiée à l'URI de demande ainsi que de la totalité des ressources identifiées par ses URI de membres internes.

Si l'une quelconque des ressources identifiées par l'URI d'un membre de la collection ne peut être détruite, alors aucun des ancêtres du membre **NE DOIT** être détruit, cela afin de maintenir la cohésion de l'espace de noms.

Tout en-tête inclus avec lDELETE **DOIT** être appliqué lors du traitement de chacune des ressources à supprimer.

Quand le traitement de la méthode `DELETE` est terminé, elle **DOIT** résulter en un espace de noms cohérent.

Si une erreur survient avec une ressource autre que celle identifiée par l'URI de demande, alors la réponse **DOIT** avoir la valeur `207` (états multiples). Les erreurs de type `424` (Echec de dépendance) **NE DEVRAIENT PAS** être dans un `207` (états multiples). Elle peuvent être mises de côté en toute sécurité parce que le client saura que les ancêtres d'une ressource n'ont pu être détruits quand le client reçoit une erreur concernant la progéniture d'un ancêtre. De plus, les erreurs 204 (Pas de contenu) **NE DEVRAIENT PAS** être retournés en `207` (états multiples). La raison de cette interdiction est que le code `204` (Pas de contenu) est le code de succès par défaut.

## 8.6.2.1 Exemple - DELETE

>>Requête

```
DELETE /container/ HTTP/1.1 Host: www.foo.bar
```

>>Réponse

```
HTTP/1.1 207 Multi-Status Content-Type: text/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.foo.bar/container/resource3</d:href>
<d:status>HTTP/1.1 423 Locked</d:status>      </d:response>
</d:multistatus>
```

Dans cet exemple, la tentative de suppression de http://www.foo.bar/container/resource3 échoue parce que la ressource est verrouillée, et aucun jeton de verrou n'a été soumis avec la requête. En conséquence, la tentative de suppression de http://www.foo.bar/container/ échoue aussi. Ainsi, le client est informé que la tentative de suppression de http://www.foo.bar/container/ a échoué puisque le parent ne peut pas être supprimé tant que ses enfants ne le sont pas. Bien qu'un en-tête `Depth` n'ait pas été inclus, une profondeur de infinity est supposée parce que la méthode est appliquée à une collection.

## 8.7 Méthode `PUT`

### 8.7.1 Méthode `PUT` pour des ressources autres que des collections

Un `PUT` réalisé sur une ressource existante remplace l'entité de réponse GET de la ressource. Les propriétés définies sur la ressource peuvent être recalculées pendant le traitement du `PUT` mais ne sont pas autrement affectées. Par exemple, si un serveur reconnait le type de contenu du corps de la requête, il peut être capable d'en extraire automatiquement les informations intéressantes à exposer en tant que propriétés.

Un `PUT` dont le résultat serait la création d'une ressource sans collection parente de portée appropriée **DOIT** échouer et retourner un code `409` (Conflit).

### 8.7.2 Méthode `PUT` appliquée à des collections

Comme défini dans dans la spécification HTTP/1.1 [RFC2068], "la méthode `PUT` "impose que l'entité incluse soit mémorisée sous l'URI de demande transmis". Puisque la soumission d'une entité représentant une collection coderait implicitement la création et la suppression de ressources, cette spécification ne définit intentionnellement aucun format de transmission pour la création de collection via la méthode `PUT`. C'est la méthode `MKCOL` qui est définie dans cette spécificaiton pour la création de collections.

Quand l'opération `PUT` crée une nouvelle ressource qui n'est pas une collection, tous ses ancêtres **DOIVENT** préalablement exister. Si tous les ancêtres n'existent pas, la méthode **DOIT** échouer et retourner un code d'état 409 (Conflit). Par exemple, si la ressource /a/b/c/d.html doit être créée et

que /a/b/c/ n'existe pas, alors la requête doit échouer.

## 8.8 La méthode COPY

La méthode `COPY` crée une copie de la ressource source identifiée par l'URI de demande, dans la ressource de destination, identifiée par l'URI qui se trouve dans l'en-tête Destination. L'en-tête Destination **DOIT** être présent. Le comportement exact de la méthode COPY dépend du type de ressource source.

Toute ressource conforme à WebDav **DOIT** supporter la méthode COPY. Toutefois, le support de la méthode `COPY` ne garantit pas qu'il soit réellement possible de copier la ressource. Par exemple, des ressources pourtant présentes sur le même serveur peuvent être contrôlées par des programmes distincts. En conséquence, il pourrait n'être pas possible de copier une ressource sur un emplacement apparemment sur le même serveur.

### 8.8.1 Méthode COPY pour des ressources HTTP/1.1

Quand la ressource source n'est pas une collection, le résultat de la méthode COPY est la création d'une nouvelle ressource à la destination dpnt l'état et le comportement correspondent le plus possible à ceux de la ressource source. Après l'invocation réussie d'un `COPY`, toutes les propriétés de la ressource source **DOIVENT** être dupliquées sur la ressource de destination, en étant toutefois sujettes à des modifications de leurs en-têtes et d'éléments XML, respectant en cela les règles de copiage des propriétés. Comme l'environnement de la destination peut être différent de celui de la source pour des raisons qui sont indépendantes des possibilités de contrôle du serveur, tel que l'absence des ressources nécessaire au bon déroulement de l'opération, il peut arriver que l'ensemble des comportements de la ressource ne puisse être copié dans la destination cible. Les altérations ultérieures de la ressource de destination ne modifieront pas la ressource source. Les altérations ultérieures de la ressource source ne modifieront pas la ressource de destination.

### 8.8.2 Méthode COPY appliquée à des propriétés

La section suivante définie comment les propriétés d'une ressource sont traitées durant une opération COPY.

Les propriétés vivantes **DEVRAIENT** être dupliquées comme propriétés vivantes en conservant au niveau de la ressource de destination le même comportement que celui qu'elles ont sur la ressource source. Si une propriété ne peut pas être copiée en conservant son comportement, alors sa valeur **DOIT** être dupliquée, octet à octet, dans une propriété morte de nom identique, sur la ressource de destination et en tenant compte des conditions précisées par l'éléments XML `propertybehavior`.

L'élément XML propertybehavior peut spécifier que les propriétés sont copiées au mieux, que toutes les propriétés vivantes sont copiées avec succès sinon la méthode doit échouer, ou qu'une liste spécifiée de propriétés vivantes doit réussir sinon la méthode doit échouer. L'élément XML propertybehavior est défini à la section 12.12.

# 8.8.3 Méthode COPY appliquée à des collections

La méthode `COPY` appliquée à une collection sans en-tête Depth **DOIT** agir en prenant comme valeur par défaut "`infinity`". Les valeurs qu'un client a le droit de soumettre dans l'en-tête `depth` de la méthode `COPY` sont "`0`" ou "`infinity`". Les serveurs conformes `DAV` **DOIVENT** supporter ces deux valeurs.

La valeur `infinity` signifie que la collection d'origine, identifiée par l'URI de demande, doit être copiée à l'emplacement spécifié par l'URI fournie dans l'en-tête `Destination`, et que toutes les ressources membres internes de la collection d'origine, sur tous ses niveaux de profondeur, doivent être récursivement copiées à l'emplacement de la ressource de destination.

Une `COPY` avec "`Depth: 0`" ne fait que signifier que la collection d'origine et ses propriétés doivent être copiés mais pas les ressources identifiées par ses URI de membres internes.

Tout en-tête spécifié dans une requête `COPY` **DOIT** être appliqué à chaque ressource copiée, pendant sa copie, à l'exception de l'en-tête `Destination`.

L'en-tête Destination ne fait que spécifier l'URI "de destination" de l'URI de demande. Quand on l'applique aux membres de la collection identifiée par l'URI de demande, la valeur de Destination est modifiée pour refléter la position relative des ressources dans la hiérarchie copiée. Ainsi : soit la resource /a/ à laquelle on applique les valeurs http://fun.com/ comme valeur d'en-tête `Host` et http://fun.com/b/ pour valeur d'en-tête Destination, alors quand la ressource http://fun.com/a/c/d est copiée, la Destination qui doit être utilisée est http://fun.com/b/c/d.

Quand l'exécution de la méthode COPY est terminée, elle **DOIT** avoir créé au niveau de la destination un espace de noms cohérent (se référer à la section 5.1 pour la définition des espaces de noms cohérents). Toutefois, si une erreur se produit pendant la copie d'une collection interne, le serveur **NE DOIT PAS** copier de ressources identifiées par les membres de cette collection (en clair, par exemple, le serveur doit sauter le sous-arbre de cette collection), parce que cela reviendrait à créer un espace de noms incohérent. Quand une erreur est ainsi détectée, l'opération de COPY **DEVRAIT** essayer de se terminer de manière à ce que le résultat soit le plus proche possible de ce qu'était la ressource source à copier (par exemple, le serveur doit essayer de copier les autres sous-arbres et leurs membres de la ressource source, ceux qui ne sont pas des descendants de la collection ayant causé le problème). Par exemple, si un COPY avec une profondeur infinity est appliqué à la collection /a/, contenant les collections /a/b/ et /a/c/, et qu'une erreur intervient lors de la copie de /a/b/, alors la méthode doit quand même essayer de copier /a/c/.

De la même manière, si une erreur arrive lors de la copie d'une ressource qui n'est pas une collection et dans le cas d'un COPY de type infinity, le serveur **DEVRAIT** essayer de terminer le traitement en rendant un résultat qui soit le plus proche possible du résultat souhaité. Si une erreur dans l'exécution de la méthode COPY survient sur une ressource autre que celle identifiée par l'URI de demande alors la réponse **DOIT** être le code 207 (multi-état).

Le code 424 (Echec de dépendance) **NE DEVRAIT PAS** être retourné dans une réponse de type 207 (multi-état) résultant de l'exécution de la méthode COPY. Les réponses peuvent être omises en toute sécurité parce que le client sera informé que la progéniture d'une ressource n'a pas pu être copiée quand il recevra l'erreur relatve aux parents. De plus, les codes 201 (Créé) et 204 (Pas de contenu) **NE DEVRAIENT PAS** être retournés dans les réponses 207 (états multiples) produites par la méthode COPY. Elles peuvent également être omises en toute sécurité parce qu'il s'agit des codes de réussite par défaut.

## 8.8.4 Méthode COPY et en-tête Overwrite

Si une des ressources copiées existe au niveau de la destination spécifiée et que l'en-tête Overwrite vaut "T" alors, avant d'exécuter la copie, le serveur **DOIT** exécuter un DELETE de la ressource de destination correspondante avec l'option "Depth: infinity". Si l'en-tête Overwrite est initialisée à "F" alors l'opération va échouer.

## 8.8.5 Codes d'états

201 (Créé) - La ressource source a été copiée avec succès. Le résultat de cette opération est la création d'une nouvelle ressource.

204 (Pas de contenu) - La ressource source a été copiée avec succès vers une ressource de destination pré-existante.

403 (Interdit) - Les URI de la source et de la destination sont les mêmes.

409 (Conflit) - Une des ressources ne peut pas être créée au niveau de la destination tant qu'une ou plusieurs collections intermédiaires n'auront pas été créées.

412 (Echec de la précondition) - Le serveur a été incapable de maintenir l'aspect vivant des propriétés listées dans l'élément XML propertybehavior ou bien l'en-tête Overwrite vaut "F" et l'état de la collection de destination n'est pas nul.

423 (Verrouillé) - La ressource de destination était verrouillée.

502 (Mauvaise passerelle) - Cela peut arriver quand la destination se trouve être sur un autre serveur et que ce dernier refuse d'accepter la ressource.

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

## 8.8.6 Exemple - COPY avec Overwrite

Dans cet exemple, la ressource http://www.ics.uci.edu/~fielding/index.html est copiée à la destination http://www.ics.uci.edu/users/f/fielding/index.html. Le code d'état 204 (Pas de contenu) indique que la ressource de destination qui existait été écrasée.

>>Requête

COPY /~fielding/index.html HTTP/1.1 Host: www.ics.uci.edu Destination: http://www.ics.uci.edu/users/f/fielding/index.html

>>Réponse

HTTP/1.1 204 No Content

## 8.8.7 Exemple - COPY sans Overwrite

L'exemple suivant montre la même opération de copie, mais en ayant mis l'en-tête Overwrite à "F." Une réponse de type 412 (Echec de précondition) est retournée parce que la ressource de destination n'était pas dans un état nul.

>>Requête

COPY /~fielding/index.html HTTP/1.1 Host: www.ics.uci.edu Destination: http://www.ics.uci.edu/users/f/fielding/index.html Overwrite: F

>>Réponse

HTTP/1.1 412 Precondition Failed

## 8.8.8 Exemple - COPY d'une collection

>>Requête

```
COPY /container/ HTTP/1.1 Host: www.foo.bar Destination:
http://www.foo.bar/othercontainer/ Depth: infinity Content-Type:
text/xml; charset="utf-8" Content-Length: xxxx

      <?xml version="1.0" encoding="utf-8" ?>
<d:propertybehavior xmlns:d="DAV:">
<d:keepalive>*</d:keepalive>       </d:propertybehavior>
```

>>Réponse

```
HTTP/1.1 207 Multi-Status Content-Type: text/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.foo.bar/othercontainer/R2/</d:href>
            <d:status>HTTP/1.1 412 Precondition Failed</d:status>
        </d:response>       </d:multistatus>
```

L'en-tête Depth est inutile car infinity est le comportement par défaut de COPY sur une collection. Dans cet exemple, la plupart des resssources, ainsi que la collection, ont été copiées avec succès. Toutefois la copie de la collection R2 a échoué, probablement à cause d'un problème lié au maintien de propriétés vivantes (cela est spécifié par l'élément XML propertybehavior). Comme il y a eu cette erreur sur R2, aucun des membres de R2 n'a été copié. Toutefois aucune erreur n'a été inscrite dans la réponse pour les membres en question à cause de la règle de minimisation des erreurs décrite à la section 8.8.3.

# 8.9 Méthode MOVE

L'opération MOVE sur une ressource non collection est logiquement équivalente à l'exécution d'une

copie (COPY), suivi par un traitement de maintenance de la cohérence, suivi d'une suppression de la source, les trois actions étant exécutées séparément. L'étape de maintenance de la cohérence permet au serveur d'exécuter les mises à jour consécutives au déplacement, comme par exemple la mise à jour de tous les URI autres que l'URI de demande qui identifie la ressource source, pour les faire pointer vers la nouvelle resssource destination. En conséquence, l'en-tête Destination **DOIT** être présent sur toutes les méthodes MOVE et **DOIT** suivre toutes les exigences de la méthode COPY, tout au moins pour la partie identique à COPY de la méthode MOVE. Toutes les ressources conformes à DAV **DOIVENT** supporter la méthode MOVE. Toutefois, la prise en charge de la méthode MOVE n'est pas une garantie que le serveur aura la capacité de déplacer une ressource source vers une destination particulière.

Par exemple, des programmes indépendants peuvent avoir en fait, sur le même serveur, le contrôle des différents ensembles de ressources. Dès lors, il se pourrait qu'il ne soit pas possible de déplacer une ressource à l'intérieur d'un espace de noms qui se trouve pourtant être sur le même serveur.

Si une ressource existe à la destination, un effet colatéral de la méthode MOVE est que la ressource destination sera SUPPRIMÉE , sous réserve des restrictions dues à l'en-tête Overwrite.

## 8.9.1 Méthode MOVE appliquée aux propriétés

Le comportement des propriétés lors de l'exécution d'un MOVE, y compris les effets de l'élément XML propertybehavior, **DOIT** être le même que celui spécifié dans la section 8.8.2.

## 8.9.2 Méthode MOVE appliquée à des collections

L'en-tête "Depth: infinity" utilisé avec la méthode MOVE est une instruction qui sert à déplacer la colllection identifiée par l'URI de demande vers l'URI spécifié dans l'en-tête Destination et précise que toutes les ressources identifiées par ses URI de membres internes, à tous les niveaux hiérarchiques, sont à déplacer vers des emplacements de la ressource de destination qui lui seront relatifs, récursivement à travers tous les niveaux de la hiérarchie de collection.

La méthode MOVE appliquée à une collection **DOIT** agir comme si l'en-tête "Depth: infinity" était utilisé sur elle. Un client **NE DOIT PAS** soumettre un en-tête Depth d'un MOVE sur une collection avec une valeur autre que "infinity" .

Tous les en-têtes inclus avec MOVE **DOIVENT** être appliqués au 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 celui défini pour COPY sur les collections.

Quand l'exécution de la méthode MOVE est terminée, elle **DOIT** avoir créé un espace de noms cohérent tant au niveau de la source que de la destination (se référer à la section 5.1 pour voir la définition de la notion d'espace de noms cohérent). Toutefois, si une erreur survient pendant le déplacement d'une collection interne, le serveur **NE DOIT PAS** déplacer les resources identifiées par les membres de la collection génératrice de l'erreur (par exemple, le serveur **DOIT** sauter le sous-arbre ayant provoqué l'erreur et passer à un autre), parce que cela créerait un espace de nom incohérent. Dans ce cas, après que l'erreur est détectée, l'opération de déplacement **DEVRAIT** essayer de terminer l'action de manière à obtenir un résultat le plus proche possible de celui espéré

initialement (par exemple, le serveur devrait continuer à essayer de déplacer les autres sous-arbres et les ressources identifiées par leurs membres qui ne sont pas les descendants d'une collection ayant provoqué une erreur). Par exemple, si un MOVE de profondeur infinie est exécuté sur la collection /a/, contenant les collections /a/b/ et /a/c/, et qu'une erreur survient dans le déplacement /a/b/, le serveur doit quand même essayer de déplacer /a/c/. Similairement, après qu'une erreur est rencontrée lors du déplacement d'une ressource autre qu'une collection et dans le cas d'un MOVE de profondeur infinie, le serveur **DEVRAIT** doit essayer de terminer l'opération demandée de manière à obtenir un résultat le plus proche possible de celui attendu par l'opération de déplacement originellement demandée.

Si une erreur se produit avec une ressource autre que celle identifiée par l'URI de demande alors la réponse **DOIT** être un code 207 (états multiples).

Le code d'état 424 (Echec de dépendance) **NE DEVRAIT PAS** être retourné dans le cas d'une réponse 207 (états multiples) produite par une méthode MOVE. Ces erreurs peuvent être omises en toute sécurité parce que le client saura que la progéniture d'une ressource ne peut pas être déplacée quand il recevra le message d'erreur concernant le parent. De plus, les réponses 201 (Créé)/204 (Pas de contenu) **NE DEVRAIENT PAS** être retournées dans une réponse 207 (états multiples) à un MOVE. Ces réponses peuvent être omises en toute sécurité parce qu'il s'agit des codes de succès par défaut.

## 8.9.3 Méthode MOVE et l'en-tête Overwrite

Si une ressource existe à l'emplacement défini comme destination et que l'en-tête Overwrite est "T" alors, avant d'exécuter le déplacement le serveur **DOIT** exécuter un DELETE de profondeur infinie de la ressource destination. Si l'en-tête Overwrite est "F" alors l'opération échouera.

## 8.9.4 Codes d'états

201 (Créé) - La ressource source a été déplacée avec succès, et une nouvelle ressource a été créée à la destination indiquée.

204 (Pas de contenu) - La ressource source a été déplacée avec succès vers une ressource destination qui existait déjà.

403 (Interdit) - Les URI de la source et de la destination sont les mêmes.

409 (Conflit) - Une ressource ne peut pas être créée à la destination tant qu'une ou plusieurs collections intermédiaires n'auront pas été créées.

412 (Echec de la précondition) - Soit le serveur a été incapable de maintenir l'aspect vivant des propriétés listées dans l'élément XML propertybehavior soit l'en-tête Overwrite vaut "F" et l'état de la ressource destination n'est pas nul.

423 (Verrouillé) - La source ou la destination était verrouillée.

502 (Mauvaise passerelle) - Cela peut survenir quand la destination est sur un autre serveur et que le serveur de la destination refuse de recevoir la ressource.

## 8.9.5 Exemple - MOVE d'une ressource autre qu'une collection

Cet exemple montre le déplacement de la ressource http://www.ics.uci.edu/~fielding/index.html vers la destination http://www.ics.uci.edu/users/f/fielding/index.html. Le contenu de la ressource de destination aurait été écrasé si la ressource de destination avait été non nulle. Dans ce cas, puisqu'il n'y avait rien dans la ressource de destination, le code de réponse retourné est 201 (Créé).

>>Requête

```
MOVE /~fielding/index.html HTTP/1.1 Host: www.ics.uci.edu
Destination: http://www.ics.uci.edu/users/f/fielding/index.html
```

>>Réponse

```
HTTP/1.1 201 Created Location:
http://www.ics.uci.edu/users/f/fielding/index.html
```

## 8.9.6 Exemple - MOVE d'une collection

>>Requête

```
MOVE /container/ HTTP/1.1 Host: www.foo.bar Destination:
http://www.foo.bar/othercontainer/ Overwrite: F If:
(<opaquelocktoken:fe184f2e-6eec-41d0-c765-01adc56e6bb4>)
(<opaquelocktoken:e454f3f3-acdc-452a-56c7-00a5c91e4b77>) Content-
Type: text/xml; charset="utf-8" Content-Length: xxxx

   <?xml version="1.0" encoding="utf-8" ?>    <d:propertybehavior
xmlns:d='DAV:'>      <d:keepalive>*</d:keepalive>
</d:propertybehavior>
```

>>Réponse

```
HTTP/1.1 207 Multi-Status Content-Type: text/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.foo.bar/othercontainer/C2/</d:href>
<d:status>HTTP/1.1 423 Locked</d:status>      </d:response>
</d:multistatus>
```

Dans cet exemple, le client a soumis un certain nombre de jetons de verrou avec sa requête. Un jeton de verrou sera nécessaire pour chaque ressource verrouillée, source et destination, quel que soit l'endroit où elle se trouve dans le domaine d'application de la méthode. Dans ce cas, le jeton de verrou approprié n'a pas été soumis pour la destination http://www.foo.bar/othercontainer/C2/. Cela signifie que la ressource /container/C2/ n'a pas pu être déplacée. Et comme il y a eu une erreur sur la copie de /container/C2/, alors aucun des membres de ce répertoire n'a été copié.  Toutefois, aucune erreur n'a été inscrite dans la réponse pour ces membres à cause de la règle de minimisation

des erreurs définies à la section 8.8.3. L'authentification de l'agent d'utilisateur est déjà intervenue via un mécanisme externe au domaine du protocole HTTP, dans une couche de transport sous-jacente.

# 8.10 Méthode LOCK

Les paragraphes qui suivent décrivent la méthode LOCK, qui est utilisée pour retirer un verrou de tout ype d'accès. Ces paragraphes sur la méthode LOCK décrivent uniquement la sémantique spécifique de la méthode LOCK et qui est indépendante du type d'accès du verrou dempandé.

Toute ressource qui supporte la méthode LOCK **DOIT**, au minimum, supporter les formats XML de demande et de réponse définis ci-après.

## 8.10.1 Fonctionnement

L'invocation de la méthode LOCK crée le verrou spécifié par l'élément XML lockinfo de l'URI de demande. Les requêtes portant sur la méthode de verrouillage **DEVRAIENT** avoir un corps de requête en XML contenant l'élément XML owner, sauf si il s'agit d'une requête de rafraîchissement. La demande LOCK peut avoir un en-tête "Timeout" (temporisateur).

Les clients **DOIVENT** supposer que les verrous peuvent disparaître arbitrairement à tout instant, sans considération de la valeur donnée dans l'en-tête Timeout. L'en-tête Timeout ne fait qu'indiquer le comportement du serveur si des "circonstances extraordinaires" ne surviennent pas. Par exemple, un administrateur peut retirer un verrou à tout instant ou le système peut s'arrêter brutalement de telle manière qu'il perde l'enregistrement de l'existence du verrou. La réponse **DOIT** contenir dans l'élément XML prop la valeur de la propriété lockdiscovery.

Afin d'indiquer le jeton de verrou associé au verrou nouvellement créé, un en-tête de réponse de type Lock-Token **DOIT** être inclus dans la réponse pour chaque demande de verrou de la requête LOCK ayant réussi. Remarquer que l'en-tête Lock-Token ne serait pas retourné en réponse à une requête de rafraîchissement de verrou parce que dans ce cas, il n'y a aucune création de nouveau verrou.

## 8.10.2 L'effet des verrous sur les propriétés et les collections

Un verrou porte l'état de la totalité de la ressource, incluant son corps et ses propriétés associées. En conséquence, un verrou sur une ressource **DOIT** aussi verrouiller les propriétés de la ressource.

Pour les collections, le verrou concerne également la possibilité d'y ajouter ou d'en détruire des membres. La nature de l'effet dépend du type de contrôle d'accès en vigueur.

## 8.10.3 Verrouillage de ressources dupliquées

Une ressource peut être rendue accessible par différents URI. Toutefois les verrous s'appliquent aux ressources, pas aux URI. De ce fait, une requête LOCK sur une ressource **NE DOIT** réussir que si elle peut être honorée par tous les URI au travers desquels la ressource est accessible.

## 8.10.4 Valeur de Depth et verrouillage

L'en-tête Depth peut être utilisé avec la méthode LOCK. Les valeurs autres que "0" (zéro) et "infinity" **NE DOIVENT PAS** être utilisées avec l'en-tête Depth sur la méthode LOCK. Toutes les ressources qui supportent la méthode LOCK **DOIVENT** supporter l'en-tête Depth.

Un en-tête Depth de valeur 0 (zéro) signifie simplement qu'il faut juste verrouiller la ressource spécifiée par l'URI de demande.

Si l'en-tête Depth est initialisé à la valeur infinity alors la ressource spécifiée par l'URI de demande ainsi que tous ses membres internes, jusqu'au plus bas niveau de la hiérarchie, doivent être verrouillés. Un résultat réussi **DOIT** retourner un seul jeton de verrou représentant toutes les ressources qui ont été verrouillées. Si un UNLOCK est exécuté avec succès sur ce jeton, toutes les ressources associées sont déverrouillées. Si le verrouillage ne peut pas être accordé à toutes les ressources, un code 409 (Conflit) **DOIT** être retourné dans le corps de la réponse contenant l'élément XML multistatus décrivant quelles ressources ont empêché le verrou d'être levé. Mais le succès partiel n'est pas une option autorisée. Soit toute la hiérarchie est verrouillée, soit aucune ressource ne l'est.

Quand aucun en-tête Depth n'est soumis avec la requête LOCK alors la valeur par défaut "Depth:infinity" est appliquée.

## 8.10.5 Interaction avec d'autres méthodes

L'interaction du `LOCK` avec différentes méthodes dépend du type de verrou. Toutefois, indépendamment du type de verrou, la suppression réussie d'une ressource avec la méthode `DELETE` **DOIT** entraîner la suppression de tous les verrous qui sont rattachés à cette ressource.

## 8.10.6 Tableau de compatibilité des verrous

Le tableau ci-dessous décrit le comportement de la méthode quand une requête de verrouillage est appliquée à une ressource.

| état courant du verrou/ requête de verrou | verrou partagé | Verrou exclusif |
|---|---|---|
| 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 interdit à un principal de demander deux fois de suite le même verrou.

L'état de verrouillage courant d'une ressource est indiqué dans la colonne la plus à gauche, et les différents cas de demandes de verrouillage sont listés dans la première rangée. A l'intersection d'une colonne et d'une rangée se trouve le résultat de la demande de verrouillage. Par exemple, si un

verrou partagé est posé sur une ressource, et qu'une demande de verrou exclusif est faite, l'entrée dans le tableau donne la valeur "faux", indiquant que le verrou **NE DOIT PAS** être accordé.

## 8.10.7 Codes d'états

200 (OK) - La demande de verrouillage a réussi et la valeur de la propriété lockdiscovery est incluse dans le corps.

412 (Échec de la précondition) - Soit le jeton de verrou inclus n'a pas pu être imposé à cette ressource, soit le serveur n'a pas pu satisfaire la demande précisée dans l'élément XML lockinfo.

423 (Verrouillé) - La ressource est verrouillée, donc la méthode a été rejetée.

## 8.10.8 Exemple - Une demande de verrouillage simple

>>Requête

```
LOCK /workspace/webdav/proposal.doc HTTP/1.1 Host:
webdav.sb.aol.com Timeout: Infinite, Second-4100000000 Content-
Type: text/xml; charset="utf-8" Content-Length: xxxx
Authorization: Digest username="ejw",
realm="ejw@webdav.sb.aol.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://www.ics.uci.edu/~ejw/contact.html</d:href>
</D:owner>    </D:lockinfo>
```

>>Réponse

```
HTTP/1.1 200 OK Content-Type: text/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://www.ics.uci.edu/~ejw/contact.html </D:href>
               </D:owner>                <D:timeout>Second-
604800</D:timeout>                <D:locktoken>
                    <D:href>opaquelocktoken:e71d4fae-5dec-22d6-
fea5-00a0c91e6be4</D:href>                </D:locktoken>
</D:activelock>      </D:lockdiscovery>    </D:prop>
```

Cet exemple montre la création réussie d'un verrou d'écriture exclusif sur la ressource http://webdav.sb.aol.com/workspace/webdav/proposal.doc. La ressource

http://www.ics.uci.edu/~ejw/contact.html contient les données d'identification du propriétaire du verrou. Le serveur a une politique de temporisation fondée sur l'activité de cette ressource, et entraîne la suppression automatique du verrou après une semaine (604 800 s). Remarquez que les champs nonce, response, et opaque n'ont pas été calculés dans l'en-tête Authorization de la requête.

## 8.10.9 Exemple - rafraîchissement d'un verrou d'écriture

>>Requête

```
LOCK /workspace/webdav/proposal.doc HTTP/1.1 Host:
webdav.sb.aol.com Timeout: Infinite, Second-4100000000 If:
(<opaquelocktoken:e71d4fae-5dec-22d6-fea5-00a0c91e6be4>)
Authorization: Digest username="ejw",
realm="ejw@webdav.sb.aol.com", nonce="...",
uri="/workspace/webdav/proposal.doc", response="...", opaque="..."
```

>>Réponse

```
HTTP/1.1 200 OK Content-Type: text/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://www.ics.uci.edu/~ejw/contact.html </D:href>
               </D:owner>                <D:timeout>Second-
604800</D:timeout>                <D:locktoken>
                    <D:href>opaquelocktoken:e71d4fae-5dec-22d6-
fea5-00a0c91e6be4</D:href>                </D:locktoken>
</D:activelock>      </D:lockdiscovery>    </D:prop>
```

Cette demande rafraîchirait le verrou en réinitialisant tous les temporisateurs. Remarquer que le client a demandé un temps infini mais que le serveur a choisi d'ignorer cette demande. Dans cet exemple, les champs nonce, response, et opaque n'ont pas été calculés dans l'en-tête Authorization de la requête.

## 8.10.10 Exemple - requête de verrou sur plusieurs ressources

>>Requête

```
LOCK /webdav/ HTTP/1.1 Host: webdav.sb.aol.com Timeout: Infinite,
Second-4100000000 Depth: infinity Content-Type: text/xml;
charset="utf-8" Content-Length: xxxx Authorization: Digest
username="ejw", realm="ejw@webdav.sb.aol.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://www.ics.uci.edu/~ejw/contact.html</D:href>
</D:owner>    </D:lockinfo>
```

>>Réponse

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

   <?xml version="1.0" encoding="utf-8" ?>    <D:multistatus
xmlns:D="DAV:">      <D:response>
<D:href>http://webdav.sb.aol.com/webdav/secret</D:href>
<D:status>HTTP/1.1 403 Forbidden</D:status>      </D:response>
     <D:response>
<D:href>http://webdav.sb.aol.com/webdav/</D:href>
<D:propstat>                <D:prop><D:lockdiscovery/></D:prop>
               <D:status>HTTP/1.1 424 Failed Dependency</D:status>
          </D:propstat>      </D:response>    </D:multistatus>
```

Cet exemple montre une demande de pose d'un verrou d'écriture exclusif sur une collection et tous ses enfants. Dans cette requête, le client a spécifié qu'il souhaite obtenir un verrou de durée infinie, si un tel verrou est disponible, et, dans le cas contraire, souhaite obtenir un temps limité à 4,1 millliards de secondes, si cela est possible. Le corps de l'entité requête contient l'information d'identification du principal faisant la demande de verrouillage, ici, il s'agit de l'URL d'une page de la Toile.

La réponse retournée concernant la ressource http://webdav.sb.aol.com/webdav/secret est 403 (Interdit). Et comme cette ressource n'a pas pu être verrouillée, aucune autre ne l'a été. Remarquer aussi que la propriété lockdiscovery de l'URI de demande a été incluse dans la réponse comme cela est exigé. Dans cet exemple, la propriété lockdiscovery est vide, ce qui signifie qu'il n'y a actuellement aucun verrou de posé sur cette ressource.

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

# 8.11 Méthode UNLOCK

La méthode UNLOCK supprime de l'URI de demande le verrou identifié par le jeton de verrou spécifié dans le champ Lock-Token de l'en-tête de la requête ainsi que de toutes les ressources incluses dans le verrou. Si toutes les ressources qui ont été verrouillées par le jeton de verrou ne peuvent être déverrouillées alors la requête UNLOCK **DOIT** échouer.

Toute ressource conforme à DAV et supportant la méthode LOCK **DOIT** supporter la méthode UNLOCK.

# 8.11.1 Exemple - UNLOCK

>>Requête

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

>>Réponse

```
HTTP/1.1 204 No Content
```

Dans cet exemple, le verrou identifié par le jeton de verrou "opaquelocktoken:a515cfa4-5da4-22e1-f5b5-00a0451e6bf7" est retiré avec succès de la ressource http://webdav.sb.aol.com/workspace/webdav/info.doc. Si ce verrou avait inclu plus d'une ressource, le verrou aurait été supprimé sur chacune d'elles. Le code d'état 204 (Pas de contenu) est utilisé au lieu du code 200 (OK) parce que l'entité réponse n'a pas de corps.

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

# 9. Les en-têtes HTTP pour la rédaction distribuée

## 9.1 En-tête DAV

DAV = "DAV" ":" "1" ["," "2"] ["," 1#extend]

Cet en-tête indique que la ressource supporte le schéma et le protocole DAV tels que spécifiés. Toute ressource conforme à DAV **DOIT** retourner l'en-tête DAV avec toute réponse à la méthode OPTIONS.

La valeur est une liste de toutes les classes conformes que la ressource supporte. Remarquez que ci-dessus, une virgule a déjà été rajoutée au 2. C'est parce qu'une ressource ne peut pas avoir une conformité de niveau 2 sans être également conforme au niveau 1. Référez-vous à la section 15 pour plus de détails. En général, toutefois, le support d'une classe de conformité ne présage pas du support d'une autre.

## 9.2 En-tête Depth

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

L'en-tête Depth est utilisé avec les méthodes qui agissent sur des ressources qui ont potentiellement des membres internes afin de leur indiquer si la méthode doit être appliquée seulement à la ressource ("Depth: 0"), à la ressource et ses enfants immédiats ("Depth: 1"), ou à la ressource et toute sa progéniture ("Depth: infinity").

L'en-tête Depth est uniquement supporté si une définition de la méthode est explicitement fournie pour le support.

Les règles suivantes représentent le comportement par défaut de toute méthode supportant l'en-tête

Depth. Une méthode peut surcharger les comportements par défaut en les redéfinissant dans sa définition.

Les méthodes qui supportent l'en-tête Depth peuvent décider de pas supporter toutes les valeurs possibles et définir, au cas par cas, le comportement de la méthode lorsque l'en-tête Depth est absent. Par exemple, la méthode MOVE ne supporte que la valeur "Depth: infinity" et si l'en-tête Depth est absent elle agit comme si la valeur "Depth: infinity" avait été spécifiée.

Les clients **NE DOIVENT PAS** considérer que les méthodes s'exécutant sur les membres de leurs arborescences le font selon un ordre ou un exécution atomique particulier à moins que la méthode n'en fournisse explicitement la garantie.

Pour son exécution, une méthode ayant un en-tête Depth exécutera le mieux possible la tâche qui lui est assignée puis retournera une réponse spécifiant ce qui a été achevé et ce qui a été impossible à réaliser.

Ainsi, par exemple, une tentative de copie d'arborescence avec la méthode COPY peut réussir sur certains membres et échouer sur d'autres.

Tout en-tête de méthode dont le comportement dépend de la valeur de l'en-tête Depth **DOIT** être appliqué à toutes les ressources se trouvant dans le champ d'application de la méthode excepté aux endroits où des comportements différents ont été clairement spécifiés. Par exemple, un en-tête If-Match verra sa valeur appliquée à chaque ressource se trouvant dans le champ de la méthode et entraînera son échec sur les ressources ne satisfaisant pas à cette précondition.

Si une ressource, source ou destination, se trouvant à l'intérieur du champ d'application de la méthode ayant un en-tête Depth est verrouillée de telle manière que la méthode ne puisse réussir, alors le jeton de verrou de cette ressource **DOIT** être soumis en même temps que la requête dans l'en-tête If de la requête .

L'en-tête Depth ne fait que spécifier le comportement de la méthode au regard de ses enfants internes. Si une ressource n'a pas d'enfant interne alors l'en-tête Depth **DOIT** être ignoré.

Remarquer bien que c'est toujours une erreur que de soumettre une valeur d'en-tête Depth qui ne soit pas autorisée par la définition de la méthode. Aussi, soumettre un "Depth: 1" avec la méthode COPY, même si la ressource n'a aucun membre interne, entraînera le retour d'un code 400 (mauvaise requête). La méthode doit échouer non pas parce que la ressource n'a pas de membre interne mais à cause de la valeur interdite de l'en-tête Depth.

# 9.3 En-tête Destination

Destination = "Destination" ":" absoluteURI

L'en-tête Destination spécifie l'URI des ressources destination des méthodes de type COPY et MOVE, qui ont besoin d'avoir deux URI en paramètres. Remarquer que la règle de production de absoluteURI est définie dans la [RFC2396].

# 9.4 En-tête If

If = "If" ":" ( 1*No-tag-list | 1*Tagged-list) No-tag-list = List Tagged-list = Resource 1*List Resource = Coded-URL List = "(" 1*(["Not"](State-token | "[" entity-tag "]")) ")" State-token = Coded-URL Coded-URL = "<" absoluteURI ">"

L'en-tête If a un objectif fonctionnel similaire à l'en-tête If-Match défini à la section 14.25 de la [RFC2068]. Cependant, l'en-tête If est sensé être utilisé avec tout URI représentant une information d'état, qu'on appelle un jeton d'état, concernant aussi bien une ressource qu'un ETag. Un exemple typique de jeton d'état est un jeton de verrou, et les jetons de verrou sont les seuls jetons d'états définis dans la présente spécification.

Toute ressource conforme à DAV **DOIT** supporter l'en-tête If.

Le but de l'en-tête If est de décrire une série de listes d'états. Si l'état de la ressource à laquelle l'en-tête est appliqué ne correspond à aucun des états spécifiés alors la requête **DOIT** échouer avec un code 412 (Echec de la précondition). Si l'un des états décrits correspond à l'état de la ressource alors la requête peut réussir.

Remarquez que la règle de production de absoluteURI est définie dans la [RFC2396].

# 9.4.1 Règle de production de No-tag-list

La règle de production de No-tag-list permet de décrire une série de jetons d'état et d'ETags. Si plusieurs No-tag-list sont utilisés alors il suffit qu'un seul d'entre eux corresponde à l'état de la ressource pour que la requête puisse continuer.

Si une méthode, de par la présence d'un en-tête Depth ou Destination, s'applique à plusieurs ressources alors la règle de produciton No-tag-list **DOIT** être appliquée à chacune des ressources sur laquelle est appliquée la méthode.

# 9.4.1.1 Exemple - en-tête IF avec un No-tag-list

If: (<locktoken:a-write-lock-token> ["Je suis une ETag"]) (["Je suis une autre ETag"])

L'en-tête précédente signifie que toute ressource présente dans le champ d'application de la méthode **DOIT,** soit être verrouillée avec le jeton de verrou spécifié et être dans l'état défini par le premier Etag "I am an ETag" soit être dans l'état identifié par le second Etag "I am another ETag". Pour être tout à fait clair, on peut imaginer l'en-tête If précédente comme étant écrite sous la forme (or (and <locktoken:a-write-lock-token> ["I am an ETag"]) (and ["I am another ETag"])).

# 9.4.2 Règle de production de Tagged-list

La règle de production `tagged-list` s'applique à la production d'une liste ne s'appliquant qu'à la ressource qui la précède. Le champ d'application de la règle commence immédiatement après la spécification de la ressource et se termine avec la spécification de la ressource suivante, si il y en a une.

Quand l'en-tête `If` est appliquée à une ressource particulière, le serveur **DOIT** chercher l'existence de `Tagged-list` pour savoir si une des ressources listées correspond à la (aux) ressource(s) opérande(s) pour la méthode en cours. Si aucune des règles de production de ressources ne correspond à la ressource courante alors l'en-tête **DOIT** être ignorée. Si l'une des règles de production de ressource correspond effectivement à la ressource considérée alors la liste de production de ressources suivante **DOIT** être appliquée à la ressource de la manière spécifiée dans la section précédente.

La même URI **NE DOIT PAS** apparaître plus d'une fois dans une règle de production d'une ressource dans une en-tête `If`.

# 9.4.2.1 Exemple - en-tête If avec Tagged-List

COPY /resource1 HTTP/1.1 Host: www.foo.bar Destination: http://www.foo.bar/resource2 If: <http://www.foo.bar/resource1> (<locktoken:a-write-lock-token> [W/"A weak ETag"]) (["strong ETag"]) <http://www.bar.bar/random>(["another strong ETag"])

Dans cet exemple, la ressource http://www.foo.bar/resource1 est copiée à la ressource de destination http://www.foo.bar/resource2. Quand la méthode est appliquée pour la première fois à http://www.foo.bar/resource1, resource1 **DOIT** être dans l'état spécifié par "(<locktoken:a-write-lock-token> [W/"A weak ETag"]) (["strong ETag"])", c'est à dire, qu'elle **DOIT** être, soit verrouillée avec un jeton de verrou de type "locktoken:a-write-lock-token" et avoir une étiquette d'entité faible W/"A weak ETag" soit avoir une étiquette d'entité forte "strong ETag".

Cela est la seule condition de succès puisque la ressource http://www.bar.bar/random ne se voit jamais appliquer la méthode (la seule autre ressource listée dans l'en-tête If) et http://www.foo.bar/resource2 n'est pas listée dans l'en-tête If.

# 9.4.3 Règle de production de not

Chaque jeton d'état ou ETag est soit courant, et par conséquent décrit l'état de la ressource, soit n'est pas courant, et ne décrit pas l'état de la ressource. L'opération booléenne qui consiste à faire correspondre un jeton d'état ou ETag avec l'état courant d'une ressource se résume à une valeur vrai ou faux. La règle de production not est utilisée pour inverser cette valeur. La portée de la règle de production not est le jeton d'état ou l'étiquette d'entité qui le suit immédiatement.

If: (Not <locktoken:write1> <locktoken:write2>)

Quand cet en-tête If est soumis avec une requête, cela implique que toutes les ressources d'opérandes **NE DOIVENT PAS** être verrouillées avec locktoken:write1 mais **DOIVENT** être verrouillées avec locktoken:write2.

# 9.4.4 Fonction Matching

Quand l'en-tête If est traité, la définition de la correspondance d'un jeton d'état ou d'une étiquette d'entité se fait comme suit :

Correspondance d'une étiquette d'entité *(Matching entity tag)* : quand l'étiquette d'entité correspond

à une étiquette d'entité associée à la ressource.

Correspondance d'un jeton d'état *(Matching state token)* : quand il y a une correspondance parfaite entre le jeton d'état dans l'en-tête If et l'un des marqueurs d'état de la ressource.

## 9.4.5 En-tête If et les proxies non conformes à DAV

Les mandataires qui ne sont pas conformes à DAV ne sauront pas traiter l'en-tête If et HTTP impose que les en-têtes incompris soient ignorés. Donc, quand on communique avec des mandataires par le protocole HTTP/1.1, l'en-tête de requête "Cache-Control: no-cache" **DOIT** être utilisé afin d'empêcher que le mandataire essaie, mal, de satisfaire la requête en utilisant son antémémoire. Avec les mandataires HTTP/1.0 l'en-tête de requête "Pragma: no-cache" **DOIT** être utilisé pour les mêmes raisons.

## 9.5 En-tête Lock-Token

Lock-Token = "Lock-Token" ":" Coded-URL

L'en-tête Lock-Token est utilisé dans les requêtes de la méthode UNLOCK pour identifier le jeton de verrou à retirer. Le jeton de verrou spécifié dans l'en-tête de requête Lock-Token **DOIT** identifier un verrou s'appliquant à une ressource reconnue de l'URI de demande (c'est à dire s'appliquer à un membre de cet URI de demande).

L'en-tête Lock-Token est utilisé dans les réponses aux requêtes de la méthode LOCK pour indiquer le jeton de verrou créé quand la requête a réussi et qu'un nouveau verrou a été créé.

## 9.6 En-tête Overwrite

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

L'en-tête Overwrite précise si le serveur doit écraser l'état d'une ressource de destination non nulle en cas de COPY ou MOVE. La valeur "F" signifie que le serveur ne doit pas exécuter l'opération COPY ou MOVE si l'état de la ressource de destination est non nul. Si l'en-tête Overwrite n'est pas inclus dans une requête COPY ou MOVE alors la ressource **DOIT** traiter la demande comme si l'en-tête Overwrite avait la valeur "T". Almors que l'en-tête Overwrite vient pour dupliquer la fonctionnalité de l'en-tête If-Match: * de HTTP/1.1, If-Match s'applique uniquement à l'URI de demande, et pas à la destination d'un COPY ou d'un MOVE.

Si un COPY ou un MOVE ne peut pas s'exécuter à cause de la valeur de l'en-tête Overwrite, la méthode **DOIT** alors échouer et retourner le code d'état 412 (Echec de la précondition).

Toute ressource conforme à DAV **DOIT** supporter l'en-tête Overwrite.

## 9.7 En-tête de réponse Status-URI

L'en-tête de réponse Status-URI peut être utilisé avec le code d'état 102 (traitement) pour

transmettre une information au client comme par exemple sur l'état d'une méthode.

Status-URI = "Status-URI" ":" *(Status-Code Coded-URL) ; Status-Code est défini en 6.1.1 de [RFC2068]

Les URI listés dans l'en-tête sont des ressources de type source qui ont été modifiées par la méthode en suspens. Le code d'état indique si la méthode a pu être appliquée à la ressource identifiée. Ainsi, par exemple, si une méthode MOVE appliquée à une collection est en suspens et qu'une réponse 102 (traitement) est retournée dans l'en-tête de réponse Status-URI, les URI inclus permettront de connaître les ressources que la méthode a tenté de déplacer et quel en a été le résultat.

# 9.8 En-tête de requête Timeout

TimeOut = "Timeout" ":" 1#TimeType TimeType = ("Second-" DAVTimeOutVal | "Infinite" | Other) DAVTimeOutVal = 1*digit

Other = "Extend" field-value ; Voir la section 4.2 de la [RFC2068]

Les clients peuvent inclure les en-têtes Timeout dans leurs requêtes LOCK. Toutefois, le serveur n'est pas obligé d'honorer voir même seulement de considérer ces en-têtes. Les en-têtes Timeout sont réservés aux requêtes portant sur la méthode LOCK : les clients **NE DOIVENT PAS** soumettre d'en-tête Timeout avec toute méthode autre que la méthode LOCK.

L'en-tête Timeout **DOIT** contenir au moins un TimeType et peut en contenir plusieurs. L'objectif d'une liste de TimeType est de pouvoir indiquer au client plusieurs valeurs et types de valeurs qui lui soient acceptables. Le client liste les entrées TimeType par ordre de préférence.

Les valeurs de la réponse à un Timeout **DOIVENT** utiliser l'une des valeurs Second, Infinite, ou un TimeType que le client a déclaré comme lui étant familier. Le serveur est en droit de supposer que le client est familier avec n'importe lequel des TimeType qui lui a été soumis dans l'en-tête Timeout.

Le TimeType "Second" spécifie le nombre de secondes qui s'écouleront entre le moment ou le verrou sera accordé par le serveur et la suppression automatique de ce verrou. La valeur du temporisateur pour le TimeType "Second" **NE DOIT PAS** être plus grande que $2^{32}$-1.

Le compteur de limite de temps **DEVRAIT** être réinitialisé à chaque fois que le propriétaire d'un verrou envoie une méthode à n'importe lequel des membres du verrou, y compris des méthodes non supportées, ou des méthodes qui ont échoué. Toutefois, le verrou **DOIT** être rafraîchi si une méthode de rafraîchissement de LOCK est reçue avec succès.

Si la limite de temps est atteinte, alors le verrou peut être perdu. Plus précisément, si le serveur souhaite récupérer les verrous en dépassement de temps, il **DEVRAIT** agir comme si il exécutait une méthode UNLOCK sur la resource utilisant le jeton du verrou en dépassement de temps. Cela pourrait être fait au titre des privilèges que lui confère son autorité en écrasement. Les journaux d'événements devraient alors être mis à jour avec l'emplacement des verrous, envoyer des notifications etc..., exactement comme cela se fait pour une demande de la méthode UNLOCK.

Les serveurs sont avertis qu'ils doivent prêter une attention toute particulière aux valeurs qui leurs sont soumises par les clients, puisqu'elles seront autant d'indications du type d'activité que le client a

l'intention d'exécuter. Par exemple, une appliquette s'exécutant dans un navigateur peut avoir besoin de verrouiller une ressource ; mais à cause de l'instabilité de l'environnement dans lequel elle s'exécute, elle peut très bien être arrêtée sans qu'aucun message d'avertissement soit émis. En conséquence, il est fortement probable que l'appliquette aura besoin d'avoir un temps limite d'exécution de petite durée de manière à ce que, si elle venait à mourrir, le verrou puisse être rapidement récupéré. Toutefois, un système de gestion de documents va probablement demander une temporisation extrêmement longue parce que son utilisateur peut prévoir de passer hors connexion.

Un client **NE DOIT PAS** se supposer qu'un verrou est perdu juste parce que le temporisateur est arrivé à expiration.

# 10. Extension des codes d'états de HTTP/1.1

Les codes d'états suivants ont été ajoutés à ceux définis dans HTTP/1.1 [RFC2068].

## 10.1 Code d'état 102 : traitement

Le code d'état 102 (Traitement) est une réponse intermédiaire utilisée pour informer le client que le serveur a accepté la totalité de la requête, mais n'a pas fini de l'exécuter. Ce code d'état **DEVRAIT** être seulement envoyé quand le serveur a de sérieuses raisons de juger que la requête prendra un temps significatif avant d'être complètement terminée. Comme consigne générale, si le serveur détermine qu'une méthode prendra plus de 20 secondes (une valeur raisonnable, quoique arbitraire) pour exécuter un traitement, alors il **DEVRAIT** retourner une réponse 102 (Traitement). Le serveur **DOIT** envoyer une réponse finale après l'accomplissement de la requête.

Le traitement des méthodes peut potentiellement prendre un certain temps, particulièrement celles qui supportent l'en-tête Depth. dans ces cas là, le client peut interrompre sa connection avec le serveur alors qu'il est en attente d'une réponse. Pour se pémunir de cela, le serveur peut retourner un code d'état 102 (Traitement) pour informer le client qu'il est toujours en train d'exécuter la méthode.

## 10.2 Code d'état 207 : états multiple

Le code d'état 207 (états multiple) fournit les états de plusieurs opérations indépendantes (référez vous à la section 11 pour plus d'informations).

## 10.3 Code d'état 422 : entité impossible à traiter

Le code d'état 422 (entité impossible à traiter) signifie que le serveur comprend le type de contenu de l'entité requête (jusque là un code 415 'type de média non supporté' est inapproprié), et la syntaxe de l'entité requête est correcte (donc un code d'état 400 'mauvaise requête' est également inapproprié) mais ce serveur a été incapable d'exécuter les instructions que le corps de la requête contient. Par exemple, ces conditions peuvent être réunies quand un corps de requête contient une instance XML bien formée (c'est à dire syntaxiquement correct) mais contenant des instructions XML erronées sémantiquement.

## 10.4 Code d'état 423 : verrouillé

Le code d'état 423 (verrouillé) signifie que la ressource source ou la destination impliquée dans une méthode est verrouillée.

## 10.5 Code d'état 424 : échec de dépendance

Le code d'état 424 (échec de dépendance) signifie que la méthode n'a pu s'appliquer correctement à la ressource parce que l'action demandée par la requête dépendait d'une autre qui échoua. Par exemple, si une commande d'une méthode PROPPATCH échoue alors, au minimum, le reste des commandes échouera également et cela produira un code 424 (échec de dépendance).

## 10.6 Code d'état 507 : mémoire insuffisante

Le code d'état 507 (mémoire insuffisante) signifie que la méthode n'a pu être exécutée correctement sur la ressource parce que le serveur est incapable de mémoriser la représentation nécessaire à l'accomplissement de la requête. Cet état est considéré comme étant momentané. Si la requête qui reçu ce code d'état était le résultat d'une action utilisateur, la requête **NE DEVRAIT PAS** être répétée tant que cela n'est pas demandé par une action spécifique de l'utilisateur.

# 11. Réponse d'états multiples

Le corps de réponse par défaut 207 (états multiples) est une entité HTTP de type text/xml ou application/xml qui contient un seul élément XML appelé multistatus, qui contient un ensemble d'éléments XML appelés response qui contiennent des suites de codes d'états 200, 300, 400, et 500 générés pendant l'invocation de la méthode. Les séries de code d'état 100 **NE DEVRAIENT PAS** être enrégistrées dans l'élément XML response.

# 12. Définitions des éléments XML

Dans la section ci-dessous, la ligne finale de chaque section fournit la déclaration du type de l'élément en utilisant le format définit dans [REC-XML]. Le champ "Valeur", quand il est présent, spécifie des restrictions additionnelles par rapport au modèle de contenu de l'élément XMl en utilisant la notation BNF (en général, il s'agit de restreindre un peu plus les valeurs possibles d'un élément XML de type PCDATA).

## 12.1 Elément XML activelock

**Nom :** `activelock` **Espace de noms :** `DAV:` **Objet :** Décrit un verrou posé sur une ressource. **Modèle de contenu :** `<!ELEMENT activelock (lockscope, locktype, depth, owner?, timeout?, locktoken?) >`

## 12.1.1 Elément XML `depth`

**Nom :** `depth` **Espace de noms :** `DAV:` **Objet :** La valeur de l'en-tête `Depth`. **Valeur :** `"0" | "1" | "infinity"` **Modèle de contenu :** `<!ELEMENT depth (#PCDATA) >`

## 12.1.2 Elément XML `locktoken`

**Nom :** `locktoken` **Espace de noms :** `DAV:` **Objet :** le jeton de verrou associé à un verrou. **Description :** Le `href` contient une ou plusieurs URI de marqueurs de verrous opaques qui se réfèrent toutes au même verrou (c'est à dire, la règle de production de l'URI `OpaqueLockToken` fournie dans la section 6.4). **Modèle de contenu :** `<!ELEMENT locktoken (href+) >`

## 12.1.3 Elément XML `timeout`

**Nom :** `timeout` **Espace de noms :** `DAV:` **Objet :** La limite de temps associée à un verrou. **Valeur :** `TimeType` ; Définie à la section 9.8 **Modèle de contenu :** `<!ELEMENT timeout (#PCDATA) >`

## 12.2 Elément XML `collection`

**Nom :** `collection` **Espace de noms :** `DAV:` **Objet :** Identifie la ressource associée comme étant une collection. La propriété `resourcetype` d'une ressource collection **DOIT** avoir cette valeur. **Modèle de contenu :** `<!ELEMENT collection EMPTY >`

## 12.3 Elément XML `href`

**Nom :** `href` **Espace de noms :** `DAV:` **Objet :** Identifie le contenu de l'élément comme étant une URI. **Valeur :** URI ; Voir la section 3.2.1 of [RFC2068] **Modèle de contenu :** `<!ELEMENT href (#PCDATA)>`

## 12.4 Elément XML link

**Nom :** `link` **Espace de noms :** `DAV:` **Objet :** Identifie la propriété comme étant un lien et contient la source et la destination de ce lien. **Description :** L'élément XML `link` est utilisé pour fournir les identifiants des ressources sources et destinations d'un ou plusieurs liens lien. Le nom de la propriété qui contient l'élément XML `link` fournit le type du lien. `Link` est un élément multivalué, aussi plusieurs liens peuvent être définis ensemble pour indiquer qu'ils ont tous le même type. Les valeurs écrites dans les éléments XML `href` qui sont dans le contenu des éléments `src` et `dst` de l'élément XML `link` **NE DOIVENT PAS** être rejetés si ils pointent vers des ressources qui n'existent pas. **Modèle de contenu :** `<!ELEMENT link (src+, dst+) >`

## 12.4.1 Elément XML `dst`

**Nom :** `dst` **Espace de noms :** `DAV:` **Objet :** Indique la destination d'un lien **Valeur :** URI **Modèle de contenu :** `<!ELEMENT dst (#PCDATA) >`

## 12.4.2 Elément XML `src`

**Nom :** `src` **Espace de noms :** `DAV:` **Objet :** Indique la source d'un lien. **Valeur :** URI **Modèle de contenu :** `<!ELEMENT src (#PCDATA) >`

## 12.5 Elément XML `lockentry`

**Nom :** `lockentry` **Espace de noms :** `DAV:` **Objet :** Définit les types des verrous qui peuvent être utilisés avec la ressource. **Modèle de contenu :** `<!ELEMENT lockentry (lockscope, locktype) >`

## 12.6 Elément XML `lockinfo`

**Nom :** `lockinfo` **Espace de noms :** `DAV:` **Objet :** L'élément XML `lockinfo` est utilisé avec la méthode `LOCK` pour spécifier le type de verrou que le client souhaite créer. **Modèle de contenu :** `<!ELEMENT lockinfo (lockscope, locktype, owner?) >`

## 12.7 Elément XML `lockscope`

**Nom :** `lockscope` **Espace de noms :** `DAV:` **Objet :** Spécifie si un verrou est de type exclusif ou partagé. **Modèle de contenu :** `<!ELEMENT lockscope (exclusive | shared) >`

## 12.7.1 Elément XML `exclusive`

**Nom :** `exclusive` **Espace de noms :** `DAV:` **Objet :** Spécifie un verrou exclusif **Modèle de contenu :** `<!ELEMENT exclusive EMPTY >`

## 12.7.2 Elément XML `shared`

**Nom :** `shared` **Espace de noms :** `DAV:` **Objet :** Spécifie un verrou partagé **Modèle de contenu :** `<!ELEMENT shared EMPTY >`

## 12.8 Elément XML `locktype`

**Nom :** `locktype` **Espace de noms :** `DAV:` **Objet :** Spécifie le type d'accès d'un verrou. Pour

l'instant, cette spécification ne définit qu'un seul type de verrou, le verrou d'écriture. **Modèle de contenu :** `<!ELEMENT locktype (write) >`

## 12.8.1 Elément XML `write`

**Nom :** `write` **Espace de noms :** `DAV:` **Objet :** Spécifie un verrou d'écriture. **Modèle de contenu :** `<!ELEMENT write EMPTY >`

## 12.9 Elément XML `multistatus`

**Nom :** `multistatus` **Espace de noms :** `DAV:` **Objet :** Regroupe les messages d'états de plusieurs réponses. **Description :** L'élément `responsedescription` au plus haut niveau est utilisé pour fournir un message général décrivant la nature de la réponse. Quand cette valeur est disponible, une application peut l'utiliser au lieu de présenter les descriptions individuelles de chaque réponse. **Modèle de contenu :** `<!ELEMENT multistatus (response+, responsedescription?) >`

## 12.9.1 Elément XML `response`

**Nom :** `response` **Espace de noms :** `DAV:` **Objet :**Contient une seule réponse décrivant le résultat de l'application d'une méthode sur une ressource et/ou ses propriétés. **Description** : Un `href` particulier **NE DOIT PAS** apparaître plus d'une fois en tant qu'enfant d'un élément XML `response` de l'élément XML `multistatus`. Cette condition est imposée dans un soucis de conserver une croissance linéaire du temps de traitement des réponses. Cela évite particulièrement d'avoir à rechercher tous les `href` dans le but de regrouper ensemble les réponses. Il n'y a cependant aucune obligation quant un quelconque ordonnancement basé sur les valeurs des `href`. **Modèle de contenu :** `<!ELEMENT response (href, ((href*, status)| (propstat+)), responsedescription?) >`

## 12.9.1.1 Elément XML `propstat`

**Nom :** `propstat` **Espace de noms :** `DAV:` **Objet :** regroupement des éléments `prop` et `status`, `propstat` est associé 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** uniquement lister les noms des propriétés auxquelles s'appliquent les résultats qui se trouvent dans l'élément `status`. **Modèle de contenu** : `<!ELEMENT propstat (prop, status, responsedescription?) >`

## 12.9.1.2 Elément XML `status`

**Nom :** `status` **Espace de noms :** `DAV:` **Objet :** Contient un seul `status-line` HTTP **Valeur :** `status-line` ; défini dans [RFC2068] **Modèle de contenu :** `<!ELEMENT status (#PCDATA) >`

## 12.9.2 Elément XML `responsedescription`

**Nom :** `responsedescription` **Espace de noms :** `DAV:` **Objet :** Contint un message qui peut être affiché à l'utilisateur expliquant la nature de la réponse. **Description :** Cet élément XML fournit une information adaptée à une représentation à l'utilisateur. **Modèle de contenu :** `<! ELEMENT responsedescription (#PCDATA) >`

## 12.10 Elément XML `owner`

**Nom :** `owner` **Espace de noms :** `DAV:` **Objet :** Fournit une information sur le principal qui retire un verrou. **Description :** L'élément XML `owner` fournit assez d'information pour qu'il soit possible, soit de contacter directement un principal (comme un numéro de téléphone par exemple ou l'URI d'une adresse email), soit de trouver le principal (comme l'URL d'une page d'accueil) propriétaire du verrou. **Modèle de contenu :** `<!ELEMENT owner ANY>`

## 12.11 Elément XML `prop`

**Nom :** `prop` **Espace de noms :** `DAV:` **Objet :** Contient les propriétés relatives à une ressource. **Description :** L'élément XML `prop` est un élément structurel générique pour les propriétés définies sur les ressources. Tous les éléments compris à l'intérieur d'un élément XML `prop` **NE DOIVENT** définir **QUE** des propriétés relatives à des ressources. Aucun autre élément ne peut être utilisé à l'intérieur d'un élément `prop`. **Modèle de contenu :** `<!ELEMENT prop ANY>`

## 12.12 Elément XML `propertybehavior`

**Nom :** `propertybehavior` **Espace de noms :** `DAV:` **Objet :** Spécifie comment des propriétés sont gérées pendant les opérations `COPY` et `MOVE`. **Description :** L'élément XML `propertybehavior` spécifie comment sont doivent être gérées certaines propriétés pendant un `COPY` ou un `MOVE`. Si cet élement XML n'est pas inclus dans le corps de la requête alors le serveur est sensé agir tel que défini dans les comportements par défaut des méthodes correspondantes. Toutes les ressources conformes à WebDav **DOIVENT** supporter l'élément XML `propertybehavior`. **Modèle de contenu :** `<!ELEMENT propertybehavior (omit | keepalive) >`

## 12.12.1 Elément XML `keepalive`

**Nom :** `keepalive` **Espace de noms :** `DAV:` **Objet :** Spécifie la manière dont des propriétés vivantes doivent être copiées ou déplacées. **Description :** Si une liste d'URI est incluse comme valeur de l'élément `keepalive`, alors les propriétés nommées **DOIVENT** être vivantes après leur copie (méthode `COPY`) ou leur déplacement (méthode `MOVE`) dans la ressource de destination. Si le contenu de l'élément XML `keepalive` est `"*"`, cela signifie que toutes les propriétés vivantes de la ressource source **DOIVENT** être vivante dans la destination. Si les conditions spécifiées par l'élément `keepalive` ne peuvent pas être satisfaites alors la méthode **DOIT** échouer et retourner le code `412` (échec de précondition). Toutes les ressources conformes à `DAV` **DOIVENT** supporter

l'élément XML keepalive dans le cadre des méthodes COPY et MOVE. **Valeur :** "*" ; seule valeur autorisée comme caractère de données de type #PCDATA **Modèle de contenu :** <! ELEMENT keepalive (#PCDATA | href+) >

## 12.12.2 Elément XML omit

**Nom :** omit **Espace de noms :** DAV: **Objet :** L'élément XML omit indique au serveur qu'il doit faire tout son possible pour copier les propriétés et qu'un échec de copie d'une propriété **NE DOIT PAS** provoquer l'échec de la méthode. **Description :** Le comportement par défaut des méthodes COPY et MOVE est de copier/déplacer toutes les propriétés ou d'échouer. Dans certains cas, comme par exemple la copie d'une ressource en utilisant un protocole comme FTP, il peut arriver qu'il ne soit pas possible de copier/déplacer les propriétés associées à la ressource copiée/déplacée. Aussi, toutes les tentatives de copie/déplacement avec FTP échoueraient toujours parce que les propriétés ne pourraient pas être transportées, même en tant que propriétés mortes. Toute les ressources conformes à DAV **DOIVENT** supporter l'élément XML omit avec les méthodes COPY/MOVE.
**Modèle de contenu :** <!ELEMENT omit EMPTY >

## 12.13 Elément XML propertyupdate

**Nom :** propertyupdate **Espace de noms :** DAV: **Objet :** Contient une requête pour modifier les propriétés d'une ressource. **Description :** Cet élément XML est un élément structurel pour encadrer les informations nécessaires à la modification des propriétés d'une ressource. Cet élément XML est multivalué. **Modèle de contenu :** <!ELEMENT propertyupdate (remove | set)+ >

## 12.13.1 Elément XML remove

**Nom :** remove **Espace de noms :** DAV: **Objet :** Liste les propriéts DAV à supprimer d'une ressource. **Description :** L'élément XML remove signale que les propriétés spécifiées dans l'élément prop doivent être supprimées. On pourra noter que demander le retrait d'une propriété qui n'existe pas n'est pas une erreur. A l'intérieur de l'élément remove, tous les éléments XML sous l'élément prop **DOIVENT** être vides, puisque seuls les noms des propriétés à retirer sont requis. **Modèle de contenu :** <!ELEMENT remove (prop) >

## 12.13.2 Elément XML set

**Nom :** set **Espace de noms :** DAV: **Objet :** Liste les valeurs des propriétés DAV qui doivent être initialisées sur une ressource. **Description :** L'élément XML set **DOIT** contenir uniquement l'élément XML prop. Les éléments contenus dans cet élément prop **DOIVENT** spécifier le nom et la valeur des propriétés qui doivent être initialisées sur la ressource identifiée par l'URI de demande. Si une propriété existe déjà alors sa valeur est remplacée. L'information sur le langage utilisé dans la valeur de la propriété (dans l'attribut "xml:lang" quand il est présent) **DOIT** être enrégistrée de manière persistente avec la propriété, et **DOIT** être subséquemment accessible par la méthode PROPFIND. **Modèle de contenu :** <!ELEMENT set (prop) >

## 12.14 Elément XML `propfind`

**Nom :** `propfind` **Espace de noms :** `DAV:` **Objet :** Spécifie les propriétés que la méthode `PROPFIND` doit retourner. Il y a deux éléments spécifiques à utiliser avec `propfind`, il s'agit de `allprop` et `propname`. Si l'élément `prop` est utilisé à l'intérieur de `propfind` il **DOIT** uniquement contenir les noms des propriétés, pas leurs valeurs. **Modèle de contenu :** `<!ELEMENT propfind (allprop | propname | prop) >`

## 12.14.1 Elément XML `allprop`

**Nom :** `allprop` **Espace de noms :** `DAV:` **Objet :** L'élément XML `allprop` spécifie que tous les noms et valeurs de propriétés sur la ressource doivent être retournés. **Modèle de contenu :** `<! ELEMENT allprop EMPTY >`

## 12.14.2 Elément XML `propname`

**Nom :** `propname` **Espace de noms :** `DAV:` **Objet :** L'élément XML `propname` spécifie que seule une liste de noms et valeurs de propriétés de la ressource doit être retournée. **Modèle de contenu :** `<!ELEMENT propname EMPTY >`

# 13. 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 section ci-dessous, la ligne finale de chaque section fournit la déclaration du type de l'élément en utilisant le format définit dans [REC-XML]. Le champ "Valeur", là où il est présent, précise les évenuelles restrictions qu'il peut y avoir sur le contenu autorisé de l'élément XML en utilisant la notation BNF (c'est à dire, pour restreindre les valeurs du contenu des éléments dont le modèle de contenu est PCDATA).

## 13.1 Propriété `creationdate`

**Nom :** `creationdate` **Espace de noms :** `DAV:` **Objet :** Enregistre l'heure et la date à laquelle la ressource a été créée. **Valeur :** `date-time` ; se référer à l'annexe 2 **Description :** La propriété `creationdate` devrait être définie pour toute ressource conforme à DAV. Quand elle est présente, elle contient le cachet de la date de création de la ressource (définie comme étant le moment à partir duquel son état n'était plus nul). **Modèle de contenu :** `<!ELEMENT creationdate (#PCDATA) >`

## 13.2 Propriété `displayname`

**Nom :** `displayname` **Espace de noms :** `DAV:` **Objet :** Contient un nom de ressource qui soit adapté à la lecture par un utilisateur. **Description :** La propriété `displayname` devrait être définie pour toute ressource conforme à DAV. Quand elle existe, la propriété contient une description de la

ressource adaptée à sa présentation à un utilisateur. **Modèle de contenu :** `<!ELEMENT displayname (#PCDATA) >`

## 13.3 Propriété `getcontentlanguage`

**Nom :** `getcontentlanguage` **Espace de noms :** `DAV:` **Objet :** Contient l'en-tête `Content-Language` retournée en réponse à un `GET` sans en-tête `accept` **Description :** La propriété `getcontentlanguage` **DOIT** être définie pour toute ressource conforme à DAV qui retourne l'en-tête `Content-Language` suite à un `GET`. **Valeur :** `language-tag` ; `language-tag` est défini à la section 14.13 de [RFC2068] **Modèle de contenu :** `<!ELEMENT getcontentlanguage (#PCDATA) >`

## 13.4 Propriété `getcontentlength`

**Nom :** `getcontentlength` **Espace de noms :** `DAV:` **Objet :** Contient l'en-tête `Content-Length` retournée en réponse à un `GET` sans en-tête `accept`. **Description :** La propriété `getcontentlength` **DOIT** être définie pour toute ressource conforme à DAV qui retourne l'en-tête `Content-Length` en réponse à un `GET`. **Valeur :** `content-length` ; Voir la section 14.14 of [RFC2068] **Modèle de contenu :** `<!ELEMENT getcontentlength (#PCDATA) >`

## 13.5 Propriété `getcontenttype`

**Nom :** `getcontenttype` **Espace de noms :** `DAV:` **Objet :** Contient l'en-tête `Content-Type` retournée en réponse à un `GET` sans en-tête `accept`. **Description :** Cette propriété `getcontenttype` **DOIT** être définie pour toute ressource conforme à DAV qui retourne l'en-tête `Content-Type` en réponse à un `GET`. **Valeur :** `media-type` ; Définie à la section 3.7 de [RFC2068] **Modèle de contenu :** `<!ELEMENT getcontenttype (#PCDATA) >`

## 13.6 Propriété `getetag`

**Nom :** `getetag` **Espace de noms :** `DAV:` **Objet :** Contient l'en-tête `ETag` retournée en réponse à un `GET` sans en-tête `accept`. **Description :** La propriété `getetag` **DOIT** être définie pour toute ressource conforme à DAV qui retourne l'en-tête `Etag`. **Valeur :** `entity-tag` ; Définie à la section 3.11 de [RFC2068] **Modèle de contenu :** `<!ELEMENT getetag (#PCDATA) >`

## 13.7 Propriété `getlastmodified`

**Nom :** `getlastmodified` **Espace de noms :** `DAV:` **Objet :** Contient l'en-tête `Last-Modified` retournée en réponse à un `GET` sans en-tête `accept`. **Description :** Remarquez que la date de dernière modification d'une ressource peut être le reflet de changements à n'importe quel endroit de la ressource, pas nécessairement un changement qui serait seulement consécutif à un `GET`. Par exemple, le changement d'une propriété peut, à lui seul, provoquer le changement de la

date de dernière modificationa. La propriété `getlastmodified` **DOIT** être définie pour toute ressource conforme à DAV qui renvoie l'en-tête `Last-Modified` en réponse à un `GET`. **Valeur :** `HTTP-date` ; Définie à la section 3.3.1 of [RFC2068] **Modèle de contenu :** `<!ELEMENT getlastmodified (#PCDATA) >`

# 13.8 Propriété `lockdiscovery`

**Nom :** `lockdiscovery` **Espace de noms :** `DAV:` **Objet :** Décrit les verrous actifs des ressources **Description :** La propriété `lockdiscovery` retourne une liste des ressources qui ont un verrou, de quel type de verrou il s'agit, le type de limite de temps et le temps restant avant expiration du temps limite, et le jeton de verrou associé. Le serveur est libre de cacher toute ou partie de cette information si le principal qui en fait la demande n'a pas les privilèges suffisants pour voir les données demandées. **Modèle de contenu :** `<!ELEMENT lockdiscovery (activelock)* >`

# 13.8.1 Exemple - Obtenir la propriété `lockdiscovery`

>>Requête

```
PROPFIND /container/ HTTP/1.1 Host: www.foo.bar Content-Length:
xxxx Content-Type: text/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: text/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.foo.bar/container/           <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>opaquelocktoken:f81de2ad-7f3d-a1b2-4f3c-
00a0c91a9d76</D:href>                                  </D:locktoken>
                        </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 exclusif en écriture, avec une limite de temps infinie.

## 13.9 Propriété `resourcetype`

**Nom :** `resourcetype` **Espace de noms :** `DAV:` **Objet :** Spécifie la nature de la ressource. **Description :** La propriété `resourcetype` **DOIT** être définie pour toute ressource conforme à DAV. La valeur par défaut est vide. **Modèle de contenu :** `<!ELEMENT resourcetype ANY >`

## 13.10 Propriété `source`

**Nom :** `source` **Espace de noms :** `DAV:` **Objet :** La propriété `source` permet de connaître la ressource qui contient la source du lien avant son traitement. **Description :** La source du lien (`src`) est typiquement l'URI de la ressource de sortie sur laquelle le lien est défini, et il n'y a, en général, qu'une seule destination (`dst`) à ce lien, qui est l'URI auquel la source non traitée de la ressource peut être accédée. Quand plus d'un lien de destination existe, cette spécification n'introduit pas de politique d'ordonnancement. **Modèle de contenu :** `<!ELEMENT source (link)* >`

## 13.10.1 Exemple - une propriété `source`

```
   <?xml version="1.0" encoding="utf-8" ?>    <D:prop
xmlns:D="DAV:" xmlns:F="http://www.foocorp.com/Project/">
<D:source>           <D:link>
<F:projfiles>Source</F:projfiles>
<D:src>http://foo.bar/program</D:src>
<D:dst>http://foo.bar/src/main.c</D:dst>           </D:link>
          <D:link>
<F:projfiles>Library</F:projfiles>
<D:src>http://foo.bar/program</D:src>
<D:dst>http://foo.bar/src/main.lib</D:dst>           </D:link>
          <D:link>
<F:projfiles>Makefile</F:projfiles>
<D:src>http://foo.bar/program</D:src>
<D:dst>http://foo.bar/src/makefile</D:dst>           </D:link>
     </D:source>    </D:prop>
```

Dans cet exemple, la ressource http://foo.bar/program a une propriété source qui contient trois liens. Chaque lien contient trois éléments, deux (src et dst) font partie du schéma DAV défini dans le présent document et l'un (projfiles) est défini dans le schéma http://www.foocorp.com/project/ (Source, Library, et Makefile). Un client qui ne mettrait en œuvre que les éléments de la spécification DAV ne comprendrait pas les éléments du schéma foocorp et les ignorerait, ne voyant ainsi que les les liens source et destination. Un client plus évolué, capable d'exploiter les éléments du schéma foocorp, serait capable de présenter à l'utilisateur plus d'informations relatives aux liens. Cet exemple démontre la puissance du balisage XML, qui permet aux valeurs d'éléments d'être exploitées ou non en fonction des capacités des clients.

# 13.11 Propriété `supportedlock`

**Nom :** supportedlock **Espace de noms :** DAV: **Objet :** Propriété utilisée pour obtenir la liste des possibilités de vérouillage supportées par la ressource. **Description :** La propriété supportedlock d'une ressource retourne une liste des combinaisons des types d'accès et de portée qui peuvent être spécifiées dans une requête de vérouillage sur une ressource. Remarquez que les contenus réels sont eux-mêmes contrôlés par des contrôles d'accès de manière à ce qu'un serveur ne soit jamais tenu de communiquer une information à un client qui ne serait pas autorisé à la recevoir. **Modèle de contenu :** <!ELEMENT supportedlock (lockentry)* >

# 13.11.1 Exemple - obtenir la propriété supportedlock

>>Requête

```
PROPFIND /container/ HTTP/1.1 Host: www.foo.bar Content-Length:
xxxx Content-Type: text/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: text/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.foo.bar/container/</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>
```

# 14. Instructions pour le traitement de XML dans DAV

Toute ressource conforme DAV **DOIT** ignorer tout élément XML inconnu et tous ses enfants rencontrés pendant le traitement d'une méthode DAV utilisant XML comme langage de commande.

Cette restriction s'applique également au traitement, par des clients, de valeurs de propriétés DAV dans lesquelles les éléments XML inconnus **DEVRAIENT** être ignorés à moins que le schéma de la propriété en décide autrement.

Cette restriction ne s'applique pas à l'initialisation de propriétés DAV mortes sur le serveur pour lesquelles le serveur **DOIT** enregistrer les éléments XML inconnus.

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

Puisque que XML peut être transporté sous les types text/xml ou application/xml, un serveur DAV **DOIT** accepter les requêtes des méthodes DAV ayant des paramètres XML déclarés de types text/xml ou application/xml, et un client DAV **DOIT** accepter des réponses XML exprimées sous la forme de types text/xml ou application/xml.

# 15. Classes de conformité à DAV

Une ressource conforme à DAV peut choisir entre deux classes de conformité. Un client peut découvrir les classes de conformité d'une ressource en exécutant la méthode OPTIONS sur la ressource, et en examinant l'en-tête "DAV" qui est alors retourné.

Puisque ce document décrit des extensions au protocole HTTP/1.1, toutes les ressources conformes à DAV, les clients et les proxies **DOIVENT** être, au minimum, conformes à [RFC2068].

Les classes de conformité ne se suivent pas nécessairement de manière linéaire. Une ressource qui est conforme à la classe 2 **DOIT** aussi être conforme à la classe 1 ; mais si des classes de conformité complémentaires sont définies ultérieurement, une ressource qui est conforme aux classes 1, 2, et 4 pourrait très bien ne pas être conforme à la classe 3. Noter aussi que les nombres ne sont pas les seuls identifiants possibles des classes de conformité.

## 15.1 Classe 1

Une ressource conforme de classe 1 **DOIT** satisfaire à toutes les exigences de toutes les sections de ce document.

Une ressource conforme de classe 1 **DOIT** retourner, au minimum, la valeur "1" dans l'en-tête DAV de réponse à la méthode OPTIONS.

## 15.2 Classe 2

Une ressource conforme de classe 2 **DOIT** satisfaire toutes les conditions de classe "1" et supporter la méthode LOCK, la propriété supportedlock, la propriété lockdiscovery, l'en-tête de réponse Time-Out et l'en-tête de requête Lock-Token. Une ressource conforme de classe "2" **DEVRAIT** aussi supporter l'en-tête de requête Time-Out et l'élément XML owner.

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

# 16. Considérations sur l'internationalisation

Dans le domaine de l'internationalisation, cette spécification est conforme à la politique de l'IETF sur les jeux de caractères [RFC2277]. Dans cette spécification, des champs lisibles par l'homme peuvent être trouvés soit sous la forme de valeurs de propriétés, soit de messages d'erreur retournés dans le corps d'une entité réponse. Dans les deux cas, le contenu humainenement lisible est codé en utilisant XML, qui donne très précisément les règles de balisage et d'encodage des jeux de caractères, et exige des processeurs XML qu'ils sachent lire des éléments XML codés, au minimum, en utilisant l'UTF-8 [UTF-8] du niveau multilangue de l'ISO 10646. Les exemples XML de cette spécification montrent l'utilisation du paramètre charset de l'en-tête Content-Type, tel que défini dans [RFC2376], tout comme l'attribut encoding de XML, qui permettent aux processeurs MIME et XML d'identifier le jeu de caractères utilisé.

XML fournit aussi la possibilité de balisage du langage permettant de spécifier le langage utilisé dans le contenu spécifique d'un élément XML. XML utilise soit les balises de langue déposées à l'IANA (se référer à [RFC1766]) soit les balises de langue de l'ISO 639 [ISO-639] dans l'attribut "xml:lang" de tout élément XML et qui permet d'en identifier la langue utilisée tant au niveau de son contenu que de ses attributs.

Les applications WebDAV **DOIVENT** supporter les balises relatives aux jeux de caractères, l'encodage des caractères, et les fonctionnalités des balises relativess aux langues de la spécification XML. Les développeurs d'applications WebDav sont fortement encouragés à lire "XML Media Types" [RFC2376] pour connaître les instructions sur l'utilisation des types MIME pour le transport des données XML, et sur l'utilisation du paramètre charset de l'en-tête Content-Type.

Les noms utilisés dans cette spécification se décomposent en trois catégories: les noms des éléments du protocole tels que ceux des méthodes et des en-têtes, les noms d'éléments XML, et les noms de propriétés. Le nommage des éléments du protocole suit ce qui a déjà été fait pour HTTP, utilisant des noms anglais codés en ASCII américain pour les méthodes et les en-têtes. A partir du moment où ces éléments du protocole sont invisibles aux utilisateurs, et ne sont, en fait, que de simples identifiants d'unités lexicales longues, ils n'ont pas besoin de supporter plusieurs types de codage de caractères. De la même manière, même si les noms des éléments XML utilisés dans cette spécification sont anglais et codés en UTF-8, ils sont invisibles pour les utilisateurs, et n'ont donc pas besoin de supporter plusieurs sortes de codage de caractères.

Le nom d'une propriété définie sur une ressource est un URI. Bien que quelques applications (par exemple, un visualiseur de propriétés générique) affiche les URI des propriétés directement aux utilisateurs, on s'attend à ce que la plupart des applications utilisent un ensemble stable de propriétés, et fournissent des correspondances entre l'URI d'un nom de propriété et un champ humainement lisible pour les besoins de l'affichage aux utilisateurs. C'est seulement dans le cas où l'ensemble des propriétés est inconnu à l'avance qu'une application ne pourra faire autrement que de présenter aux utilisateurs les URI des noms des propriétés. Nous recommandons que les applications fournissent, chaque fois que cela est possible, des noms de propriétés lisibles par l'homme.

Pour les comptes-rendus d'erreurs, nous suivons la convention des codes d'état de HTTP/1.1, incluant avec chaque code d'état une brève description en français du code (par exemple 423 (Verrouillé)). Alors qu'il est possible qu'un programme utilisateur n'ayant pas un développement poussé se contente d'afficher ce message à l'utilisateur, les applications internationales le remplaceront par un message circonstancié, traduit dans la langue natale de l'utilisateur et en utilisant le jeu de caractère adéquat.

Puisque les traitements entre les serveurs et les clients n'exigent pas que l'information soit localisée, cette spécification ne précise aucun mécanisme de transmission d'une telle information.

# 17. Considérations sur la sécurité

Cette section contient des considérations détaillées quant à la sécurité et leurs conséquences sur les applications WebDav développées.

Toutes les considérations sur la sécurité de HTTP/1.1 (discutées dans [RFC2068]) et XML (discutées dans [RFC2376]) s'appliquent également à WebDAV. De plus, les risques inhérents à la rédaction à distance exigent une technologie d'authentification plus forte, introduisent quelques nouveaux problèmes de confidentialité, et peuvent accroître les risques avec les serveurs mal conçus. Ces considérations sont exposées ci-dessous.

## 17.1 Authentification des clients

Par leur caractéristique principale qui est d'assurer des fonctions orientées sur la rédaction, les serveurs WebDav ont besoin d'utiliser des techniques d'authentification pour protéger non seulement l'accès a une ressource quelconque du réseau , mais également l'intégrité de cette même ressource. De plus, l'introduction des fonctions de verrouillage exige de supporter les mécanismes d'authentification.

Un mot de passe envoyé sur un canal non sécurisé et "en clair" est un moyen inadapté d'assurer la sécurité d'accès et d'intégrité d'une ressource puisque ce mot de passe pourrait être intercepté. Puisque l'authentification de base d'HTTP/1.1 effectue essentiellement une transmission en clair du mot de passe, une simple authentification **NE DOIT PAS** être utilisée pour authentifier un client WebDav à un server sauf en cas de connexion sécurisée. De plus, un serveur WebDav **NE DOIT PAS** envoyer de simples certificats d'authentification dans un en-tête d'authentification WWW à moins que la connexion ne soit sécurisée. Des exemples de connexions sécurisées sont les systèmes "Transport Layer Security (TLS)" employant une série de chiffres forts avec authentification mutuelle du client et du serveur, ou une connexion sur un réseau qui est physiquement sécurisé, par exemple, un réseau isolé dans un immeuble avec des droits d'accès restreints.

Les applications WebDAV **DOIVENT** supporter le schéma d'authentification par résumé [RFC2069]. Puisque le système d'authentification par résumé vérifie que les deux parties d'une communication connaissent un secret partagé, un mot de passe, sans avoir à l'échanger en clair, l'authentification par résumé évite les problèmes de sécurité inhérents à l'authentification simple tout en fournissant un niveau d'authentification qui soit utile dans un grand nombre de cas.

## 17.2 Déni de service

Les attaques par déni de service doivent être un sujet d'attention particulier pour les serveurs WebDav. WebDAV avec HTTP permet ce genre d'attaques à plusieurs niveaux des ressources du système.

La mémoire sous-jacente peut être attaquée en déposant par PUT des fichiers extrêmement grands.

Demander des opérations récurrentes sur des grandes collections peut être une attaque contre les

temps de traitement afin de dégrader les performances.

Construire plusieurs requêtes en parallèle sur plusieurs connexions peut être une attaque contre les connexions du réseau.

Les serveurs WebDAV ont besoin d'être protégés à tous les niveaux contre ces tentatives de déni de service.

## 17.3 La sécurité par l'obscurité

WebDAV fournit, au travers de la méthode PROPFIND, un mécanisme permettant de lister les ressources membres d'une collection. Cela diminue grandement l'efficacité des techniques de sécurité et de confidentialité qui s'appuient seulement sur la difficulté à trouver le nom des ressources réseau. Les utilisateurs de serveurs WebDav sont encouragés à utiliser des techniques de contrôle d'accès afin de prévenir tout accès indésirable aux ressources, plutôt que de dépendre de la relative obscurité de leurs noms de ressources.

## 17.4 Considérations sur la confidentialité par rapport aux verrous

Quand on soumet une requête de verrouillage, un agent d'utilisateur peut aussi soumettre un champ XML owner qui contient l'information de contact de la personne qui retirera le verrou (pour les cas où une personne, plutôt qu'un robot, retirerait le verrou). Cette information de contact est enregistrée dans la propriété lockdiscovery de la ressource, et peut être utilisée par d'autres collaborateurs pour entamer une négociation d'accès à la ressource. Toutefois, dans de nombreux cas, cette information de contact peut et doit rester confidentielle et ne devfrait pas être divulguée. Les serveurs **DEVRAIENT** limiter l'accès en lecture à la propriété lockdiscovery que dans des cas appropriés. De plus, les agents d'utilisateur **DEVRAIENT** fournir un moyen permettant de contrôler si l'information de contact doit être envoyée ou pas, et que si elle est envoyée, contrôler exactement ce qui est envoyé.

## 17.5 Considérations sur la confidentialité par rapport aux propriétés

Puisque les valeurs de propriétés sont typiquement utilisées pour contenir des informations telles que le nom de l'auteur du document, il y a là une possibilité pour que des problèmes de confidentialité apparaîssent à cause des possibilités d'accès extrêmement larges aux données des propriétés des ressources. Pour réduire le risque de divulgation intempestive d'information privée via les propriétés, les serveurs sont encouragés à développer des mécanismes de contrôle d'accès qui séparent les droits d'accès en lecture au corps de la ressource de ceux de lecture de ses propriétés. Cela permet à un utilisateur de contrôler la dissemination des données des propriétés sans pour autant trop réduire les droits d'accès à la ressource elle-même.

## 17.6 Réduction de la sécurité à cause de la liaison de source

HTTP/1.1 lance un avertissement contre la fourniture de droits d'accès en lecture à du code de

descriptifs parce qu'ils peuvent contenir des informations sensibles. D'ores et déjà, WebDAV, au travers de sa fonction de liaison de source, peut potentiellement fournir un URI pour des ressources de descriptif pour qu'elles puissent être publiées. Avec HTTP/1.1, un serveur pouvait raisonnablement protéger l'accès aux ressources sources à cause de la prédominence des accès de type "lecture seule". WebDAV, qui met l'accent sur la publication, encourage les accès en lecture/écriture sur les ressources, et fournit la fonction de liaison de source pour identifier la source. Cela réduit les bénéfices pour la sécurité d'empêcher l'accès aux ressources source. Les utilisateurs et les administrateurs des serveurs WebDav doivent être très vigilants quand ils autoriseront la publication à distance de descriptifs, limitant les accès en lecture et écriture aux ressources source aux seuls principaux autorisés.

## 17.7 Conséquences de l'usage d'entités XML externes

XML supporte une fonctionalité connue sous le nom "entités externes", définie à la section 4.2.2 de [REC-XML], qui signifie au processeur XML d'aller chercher et d'inclure un fragment XML se trouvant à un URI particulier. Une entité XML externe peut être utilisée pour ajouter un morceau ou modifier une Déclaration de Type de Document (DTD) associée à un document XML. Une entité XML externe peut également être utilisée pour inclure un contenu XML à l'intérieur d'un document XML. En ce qui concernen les fragments XML non validants, comme par exemple le XML utilisé dans cette spécification, l'inclusion d'entités XML externes n'est pas une exigence de [REC-XML]. Toutefois, [REC-XML] dit clairement qu'un processeur XML peut, à sa discrétion, inclure l'entité XML externe.

Les entités XML externes n'ont pas de fiabilité intrinsèque et sont sujettes à toutes les attaques qui sont endémiques à toute requête GET HTTP. Plus encore, une entité XML externe peut modifier la DTD, et éventuellement affecter la forme finale d'un document XML, dans le pire des cas, elle peut modifier de façon significative sa sémantique, ou exposer le processeur XML aux risques concernant la sécurité discutés dans la [RFC2376]. Par conséquent, les développeurs doivent être avertis que les entités XML externes qui peuvent être traitées ne sont pas fiables.

Il y a également un risque d'adaptabilité qui accompagnerait le large déploiement d'une application qui utiliserait les entités XML externes. Dans ce cas, il est possible qu'il puisse y avoir un nombre significatif de requêtes pour une seule et même entité XML externe, cela pouvant potentiellement surcharger tout serveur génèrant des requêtes pour la ressource contenant l'entité XML externe.

## 17.8 Risques en relation avec les jetons de verrou

Cette spécification, dans sa section 6.4, exige l'utilisation des identifiants universels uniques (UUID, *Universal Unique Identifier*) pour les jetons de verrou, afin de garantir leur unicité dans le temps et l'espace. Les UUID, tels que définis dans [ISO-11578], contiennent un champ "node" qui "consiste en l'adresse IEEE, qui d'habitude est celle de l'ordinateur. Pour les systèmes qui ont plusieurs noeuds IEEE 802, toute adresse de nœud disponible peut être utilisée." Puisqu'un serveur WebDav produira beaucoup de verrous au cours de sa vie, la conséquence est qu'il va aussi publiquement exposer ses addresses IEEE 802.

Il y a plusieurs risques consécutifs à l'exposition d'adresses IEEE 802. En utilisant les adresses IEEE 802 :

- Il est possible de suivre le mouvement du matériel de sous-réseaux en sous-réseaux.

- Il peut être possible d'identifier le fabricant du matériel exécutant un serveur WebDav.

- Il peut être possible de déterminer le nombre de chaque type d'ordinateur exécutant WebDav.

Dans la section 6.4.1 de cette spécification, on y détaille un mécanisme de remplacement pour générer le champs "node" d'un UUID sans utiliser une adresse IEEE 802, ce qui allège les risques associés à l'exposition des adresses IEEE 802 par utilisation d'une autre source d'unicité.

# 18. Considérations pour l'IANA

Ce document définit deux espaces de noms, l'un s'applique aux noms de propriétés, l'autre aux éléments XML spécifiques de WebDav utilisés pour les valeurs des propriétés.

Des URI sont utilisés dans les deux cas de figures et cela pour plusieurs raisons. L'usage d'un URI n'impose aucune démarche administrative de dépôt auprès d'un organisme officiel de nommage faisant autorité, donc cela permet aux utilisateurs de définir rapidement les noms des propriétés WebDav et des éléments XML utilisés par eux-mêmes ou des applications de WebDav. Les URI fournissent également un espace d'adressage unique garantissant que les utilisateurs répartis de WebDav n'auront aucune collision entre des noms de propriétés ou d'éléments XML qu'ils auront créés.

Cette spécification définit des ensembles distincts pour les noms de propriétés et les noms d'éléments XML qui sont compris par toutes les applications WebDav. Les noms des propriétés et des éléments XML de cette spécification sont tous dérivés de l'URI de base DAV: et en lui ajoutant un suffixe : par exemple "DAV:creationdate" pour la propriété "creationdate".

Cette spécification définit également un plan d'URI pour le codage des jetons de verrou : le plan d'URI opaquelocktoken décrit dans la section 6.4.

Pour garantir une interopérabilité correcte sur la base de cette spécification, l'IANA doit réserver les espaces de noms "DAV:" et "opaquelocktoken:" à l'usage de cette spécification, ses révisions, et des spécifications relatives à WebDav.

# 19. Propriété intellectuelle

La notice suivante a été copiée de la [RFC2026], section 10.4, et décrit la position de l'IETF concernant les réclamations de propriété intellectuelle qui pourraient concerner ce document.

L'IETF ne prend aucune position quant à la validité ou l'étendue d'une quelconque propriété intellectuelle ou autres droits qui pourraient être réclamés pour son implémentation ou utiliser d'autre technologie décrite dans ce document ou l'étendue à laquelle toute licence sous de tels droits pourrait ou ne pourrait pas être disponible ; ni même représente qu'un quelconque effort n'ait été fait pour identifier l'existence de tels droits. Les informations sur les procédures de l'IETF relatives au droits sur la documentation des standards eux mêmes ou leur documentation dérivée peuvent être trouvée dans le BCP-11. Les copies des réclamations de droits applicables aux publications ainsi que toute assurance de licences à rendre disponible, ou le résultat d'une tentative faite pour obtenir une licence générale ou la permission de pouvoir utiliser de tels droits de propriétés par les intégrateurs ou les utilisateurs de cette spécification peuvent être obtenus au secrétariat de l'IETF.

L'IETF invite toute personne intéressée à porter à son attention tous les droits d'auteurs, brevets ou applications de brevet, ou tout autre droit de propriété qui pourrait couvrir la technologie qui pourrait être requise pour mettre en pratique ce standard. Nous vous remercions dans ce cas d'adresser cette information au directeur exécutif de l'IETF.

# 20. Remerciements

La réussite d'une spécification telle que celle-ci repose sur les relectures attentives et critiques et dépérit en cas d'indifférence par négligence. Les auteurs expriment leur grande reconnaissance aux personnes qui ont contribué à son élaboration et dont les noms suivent, dont la perspicacité fut tellement appréciable à chaque étape de notre travail.

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.

Nous réservons une mention spéciale à deux personnes parmi cette liste. La contribution de Larry Masinter a été incalculable, à la fois en aidant à la création de ce groupe de travail et en guidant patiemment les auteurs tout au long du chemin. Il a fixé la barre si haut que cela nous aurait fait de la peine de ne pas le satisfaire. La contribution de Judith Slein qui a porté sur la clarification de l'expression de besoins, et dans sa relecture patiente, brouillon après brouillon. Les deux ont ainsi sensiblement contrinué à l'amélioration de cette spécification et nous ont permis d'augmenter nos connaissances en gestion de document.

Nous aimerions également remercier John Turner pour le développement de la DTD XML.

# 21. Références

## 21.1 Références normatives

[RFC1766] Alvestrand, H., « Tags for the Identification of Languages », RFC 1766, March 1995.

[RFC2277] Alvestrand, H., « IETF Policy on Character Sets and Languages », BCP 18, RFC 2277, January 1998.

[RFC2119] Bradner, S., « Key words for use in RFCs to Indicate Requirement Levels », BCP 14, RFC 2119, March 1997.

[RFC2396] Berners-Lee,T., Fielding, R. and L. Masinter, « Uniform Resource Identifiers (URI): Generic Syntax », RFC 2396, August 1998.

[REC-XML] T. Bray, J. Paoli, C. M. Sperberg-McQueen, « Extensible Markup Language (XML). » World Wide Web Consortium Recommendation REC-xml-19980210. Http://www.w3.org/TR/1998/REC-xml-19980210.

[REC-XML-NAMES] T. Bray, D. Hollander, A. Layman, « Namespaces in XML ». World Wide Web Consortium Recommendation REC- xml-names-19990114. Http://www.w3.org/TR/1999/REC- xml-names-19990114/

[RFC2069] Franks, J., Hallam-Baker, P., Hostetler, J., Leach, P, Luotonen, A., Sink, E. and L. Stewart, « An Extension to HTTP : Digest Access Authentication », January 1997.

[RFC2068] Fielding, R., Gettys, J., Mogul, J., Frystyk, H. and T. Berners-Lee, « Hypertext Transfer Protocol – HTTP/1.1 », RFC 2068, January 1997.

[ISO-639] ISO (International Organization for Standardization). ISO 639 :1988. « Code for the representation of names of languages. »

[ISO-8601] ISO (International Organization for Standardization). ISO 8601 :1988. « Data elements and interchange formats – Information interchange – Representation of dates and times. »

[ISO-11578] ISO (International Organization for Standardization). ISO/IEC 11578 :1996. « Information technology – Open Systems Interconnection – Remote Procedure Call (RPC) »

[RFC2141] Moats, R., « URN Syntax », RFC 2141, May 1997.

[UTF-8] Yergeau, F., « UTF-8, a transformation format of Unicode and ISO 10646 », RFC 2279, January 1998.

## 21.2 Références informatives

[RFC2026] Bradner, S., « The Internet Standards Process – Revision 3 », BCP 9, RFC 2026, October 1996.

[RFC1807] Lasher, R. and D. Cohen, « A Format for Bibliographic Records », RFC 1807, June 1995.

[WF]C. Lagoze, « The Warwick Framework: A Container Architecture for Diverse Sets of Metadata », D-Lib Magazine, July/August 1996. http://www.dlib.org/dlib/july96/lagoze/07lagoze.html

[USMARC] Network Development and MARC Standards Office, ed. 1994. « USMARC Format for Bibliographic Data », 1994. Washington, DC : Cataloging Distribution Service, Library of Congress.

[REC-PICS] J. Miller, T. Krauskopf, P. Resnick, W. Treese, « PICS Label Distribution Label Syntax and Communication Protocols » Version 1.1, World Wide Web Consortium Recommendation REC-PICS-labels-961031. Http://www.w3.org/pub/WWW/TR/REC-PICS-labels-961031.html.

[RFC2291] Slein, J., Vitali, F., Whitehead, E. and D. Durand, « Requirements for Distributed

Authoring and Versioning Protocol for the World Wide Web », RFC 2291, February 1998.

[RFC2413] Weibel, S., Kunze, J., Lagoze, C. and M. Wolf, « Dublin Core Metadata for Resource Discovery », September 1998.

[RFC2376] Whitehead, E. and M. Murata, « XML Media Types », RFC 2376, July 1998.

# 22. Adresse des auteurs

## Y Y. Goland

Microsoft Corporation One Microsoft Way Redmond, WA 98052-6399

EMail: yarong@microsoft.com

## E J. Whitehead, Jr.

Dept. Of Information and Computer Science University of California, Irvine Irvine, CA 92697-3425

EMail: ejw@ics.uci.edu

## A Faizi

Netscape 685 East Middlefield Road Mountain View, CA 94043

EMail: asad@netscape.com

## S R. Carter

Novell 1555 N. Technology Way M/S ORM F111 Orem, UT 84097-2399

EMail: srcarter@novell.com

## D Jensen

Novell 1555 N. Technology Way M/S ORM F111 Orem, UT 84097-2399

EMail: dcjensen@novell.com

# 23. Annexes

## 23.1 Annexe 1 - DTD de WebDav

Cette section fournit une Définition de Type de Documents, correspondant aux règles de [REC-XML], pour les éléments XML utilisés dans le flot du protocole et dans les valeurs des propriétés. Elle rassemble les définitions d'éléments données aux sections 12 et 13.

```
<!DOCTYPE webdav-1.0 [    <!-------------- XML Elements from
Section 12 ------------------> <!ELEMENT activelock (lockscope,
locktype, depth, owner?, timeout?,    locktoken?) > <!ELEMENT
lockentry (lockscope, locktype) > <!ELEMENT lockinfo (lockscope,
locktype, owner?) > <!ELEMENT locktype (write) > <!ELEMENT write
EMPTY > <!ELEMENT lockscope (exclusive | shared) > <!ELEMENT
exclusive EMPTY > <!ELEMENT shared EMPTY > <!ELEMENT depth
(#PCDATA) > <!ELEMENT owner ANY > <!ELEMENT timeout (#PCDATA) > <!
ELEMENT locktoken (href+) > <!ELEMENT href (#PCDATA) > <!ELEMENT
link (src+, dst+) > <!ELEMENT dst (#PCDATA) > <!ELEMENT src
(#PCDATA) > <!ELEMENT multistatus (response+,
responsedescription?) > <!ELEMENT response (href, ((href*,
status)|(propstat+)),    responsedescription?) > <!ELEMENT status
(#PCDATA) > <!ELEMENT propstat (prop, status,
responsedescription?) > <!ELEMENT responsedescription (#PCDATA) >

   <!ELEMENT prop ANY > <!ELEMENT propertybehavior (omit |
keepalive) > <!ELEMENT omit EMPTY > <!ELEMENT keepalive (#PCDATA |
href+) > <!ELEMENT propertyupdate (remove | set)+ > <!ELEMENT
remove (prop) > <!ELEMENT set (prop) > <!ELEMENT propfind (allprop
| propname | prop) > <!ELEMENT allprop EMPTY > <!ELEMENT propname
EMPTY > <!ELEMENT collection EMPTY >    <!------------ Property
Elements from Section 13 ---------------> <!ELEMENT creationdate
(#PCDATA) > <!ELEMENT displayname (#PCDATA) > <!ELEMENT
getcontentlanguage (#PCDATA) > <!ELEMENT getcontentlength
(#PCDATA) > <!ELEMENT getcontenttype (#PCDATA) > <!ELEMENT getetag
(#PCDATA) > <!ELEMENT getlastmodified (#PCDATA) > <!ELEMENT
lockdiscovery (activelock)* > <!ELEMENT resourcetype ANY > <!
ELEMENT source (link)* > <!ELEMENT supportedlock (lockentry)* >
]>
```

## 23.2 Annexe 2 - Profils de date et de jour ISO 8601

La propriété `creationdate` précise que l'ISO 8601 [ISO-8601] doit être utilisée pour formater la date . Cette section définit un profile de format de date de l'ISO 8601 pour une utilisation avec cette spécification. Ce profile est issu d'un brouillon Internet écrit par Chris Newman, qui est mentionné ici pour que son travail lui soit correctment attribué.

```
   date-time = full-date "T" full-time    full-date = date-
fullyear "-" date-month "-" date-mday    full-time = partial-time
time-offset    date-fullyear = 4DIGIT    date-month = 2DIGIT  ;
```

```
01-12    date-mday = 2DIGIT  ; 01-28, 01-29, 01-30, 01-31 basé sur
month/year    time-hour = 2DIGIT  ; 00-23    time-minute = 2DIGIT
; 00-59    time-second = 2DIGIT  ; 00-59, 00-60 basé sur des "leap
second rules"    time-secfrac = "." 1*DIGIT    time-numoffset =
("+" / "-") time-hour ":" time-minute    time-offset = "Z" / time-
numoffset    partial-time = time-hour ":" time-minute ":" time-
second [time-secfrac]
```

Les décalages numériques sont calculés à partir de l'heure local moins l'UTC (Coordinated Universal Time). Aussi, l'heure équivalente en UTC peut être déterminée en soustrayant le décalage à l'heure locale. Par exemple, 18:50:00- 04:00 est la même chose que 22:58:00Z.

Si l'heure UTC est connue, mais que le décalage avec les heures locales ne l'est pas, cela peut être représenté avec undécalage de "-00:00". Cela est différent d'un décalage de "`Z`" impliquant que l'UTC est le point de référence préféré pour l'heure spécifiée.

# 23.3 Annexe 3 - Remarques sur le traitement des éléments XML

## 23.3.1 Remarques sur les éléments XML vides

XML supporte deux mécanismes pour indiquer que des éléments XML n'ont aucun contenu. Le premier consiste à déclarer un élément de telle sorte que son instanciation puisse être de la forme `<A></A>`. La deuxième consiste à déclarer un élément de type `EMPTY` qui s'utilisera sous la forme `<A/>`. Ces deux écritures d'éléments XML sont sémantiquement identiques. C'est une violation de la spécification XML que d'utiliser l'écriture de la forme `<A></A>` pour un élément déclaré `EMPTY` dans la DTD (c'est à dire `<!ELEMENT A EMPTY>`). Si une telle déclaration est présente dans la DTD, la forme `<A/>` est **la seule** autorisée. Si l'élément n'est pas déclaré avec un modèle de contenu `EMPTY`, alors n'importe laquelle des deux formes `<A></A>` ou `<A/>` est autorisée pour les instances vides de cet élément.

## 23.3.2 Remarques sur les traitements XML interdits

XML est un format de données flexible qui rend facile la soumission de données sous une forme qui semble apparement légale mais qui ne l'est pas en réalité. La "philosophie" du "Soit souple dans ce que tu acceptes mais strict dans ce que tu envoies" s'applique toujours, mais il **NE DOIT PAS** être appliquée de manière inappropriée. XML est extrêmement souple dans sa manière de considérer les espaces blancs, l'ordre des éléments, l'insertion de nouveaux éléments, etc... Cette souplesse n'impose de faire appel à aucune extension, particulièrement dans le domaine de la signification des éléments.

Il n'y a pas de gentillesse particulière à accepter des combinaisons illégales d'éléments XML. Au mieux, cela provoquera un résultat indésirable et au pire cela pourrait provoquer de réels dégâts.

# 23.3.2.1 Exemple - Une erreur de syntaxe XML

Le corps de requête suivante d'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` ne permet que d'utiliser l'un des éléments `allprop` ou `propname`, pas les deux en même temps. Aussi, le fragment ci-dessus est une erreur et **DOIT** provoquer en réponse l'émission du code d'erreur `400` (mauvaise requête).

Imaginez, toutefois, qu'un serveur souhaite être "sympathique" et décide de prendre l'élément `allprop` comme l'élément vrai de la requête et réponde. Un client travaillant avec une petite bande passante limitée et qui essaierait d'exécuter un `propname` serait très surpris de découvrir que le serveur a traité sa demande comme si il s'agissait d'un `allprop`.

De plus, si un serveur était indulgent et décidait de répondre à cette requête, les résultats varieraient de manière aléatoire d'un serveur à l'autre, les uns exécutants la directive `allprop`, et les autres exécutants la directive `propname`. Cela réduirait l'interopérabilité plutôt que cela ne l'augmenterait.

# 23.3.2.2 Exemple - Elément XML inconnu

L'exemple précédent était illégal parce qu'il contenait deux éléments dont il était explcitiement spécifié qu'ils s'excluaient mutuellement. Toutefois, XML est un langage extensible, donc on peut imaginer que de nouveaux éléments peuvent être définis pour une utilisation avec `propfind`. Ci-dessous se trouve le corps de la requête d'un `PROPFIND` et, comme dans l'exemple précédent, **DOIT** être rejetée avec un code de retour `400` (mauvaise requête) 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.foo.bar/standards/props/">
<E:expired-props/>    </D:propfind>
```

Pour comprendre pourquoi un code de retour `400` (mauvaise requête) est retourné, il faut regarder le corps de la requête et comment le serveur, non-familier de l'élément `expired-props`, le considère.

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

Comme le serveur ne comprend pas l'élément `expired-props`, au sens des règles de traitement XML spécifiques à WEBDAV décrites dans la section 14, il doit l'ignorer. Ainsi, le serveur voit un élément `propfind` vide, qui, par définition de l'élément `propfind` est interdit.

Vous pouvez remarquez que si l'extension avait été ajoutée cela n'aura pas pour autant forcément provoqué l'émission du code de retour `400` (mauvaise requête). Par exemple, imaginez que le corps

de requête suivant soit utilisé dans un PROPFIND:

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

L'exemple précédent contient un élément fictif leave-out. Son but est de prévoir le retour de toute propriété dont le nom correspondrait au motif soumis. Si l'exemple précédent était soumis à un serveur ne connaissant pas l'élément leave-out, le seul résultat obtenu serait que l'élément leave-out aurait été ignoré et qu'un propname aurait été exécuté.

# 23.4 Annexe 4 -- Les espaces de noms XML pour WebDAV

## 23.4.1 Introduction

Tout système conforme à DAV **DOIT** supporter les extensions d'espaces de noms XML en conformité avec la spécification [REC-XML-NAMES].

## 23.4.2 Signification des noms qualifiés

[Remarque au lecteur : Cette section n'existe pas dans [REC-XML-NAMES], mais elle est nécessaire pour éviter l'ambiguïté avec les processeurs WebDav.]

Les programmes de traitement conformes à WebDAV **DOIVENT** interpréter un nom qualifié comme un URI construit en rajoutant la LocalPart à la fin de l'URI du nom de l'espace de noms.

Exemple :

```
   <del:glider xmlns:del="http://www.del.jensen.org/">
<del:glidername>Johnny Updraft</del:glidername>
<del:glideraccidents/>    </del:glider>
```

Dans cette exemple, le nom de l'élément qualifié "del:glider" est interprété comme étant l'URL "http://www.del.jensen.org/glider".

```
   <bar:glider xmlns:bar="http://www.del.jensen.org/">
<bar:glidername>Johnny Updraft</bar:glidername>
<bar:glideraccidents/>    </bar:glider>
```

Même si cet exemple est syntaxiquement différent du précédent, il est sémantiquement égal. Chaque instance du nom de l'espace de nom "bar" est remplacée par "http://www.del.jensen.org/" qui est rajouté au nom local de chaque nom d'élément. Les noms de balises résultants dans cet exemple seront exactement les mêmes que dans l'exemple précédent.

```
   <foo:r xmlns:foo="http://www.del.jensen.org/glide">
<foo:rname>Johnny Updraft</foo:rname>      <foo:raccidents/>
```

```
</foo:r>
```

Cet exemple est également sémantiquement égal aux deux précédents. Chaque instance du nom de l'espace de noms "foo" est remplacée par l'URL "http://www.del.jensen.org/glide" qui est ensuite ajouté au nom local de chaque nom de balise, les noms de balises résultants sont identiques à ceux des exemples précédents.

# 24. Notice complète de droits de reproduction



---