Groupe de travail Réseau
Request for Comments : 4122
Catégorie : En cours de normalisation
Traduction Claude Brière de L'Isle

P. Leach, Microsoft
M. Mealling, Refactored Networks, LLC
R. Salz, DataPower Technology, Inc.

juillet 2005

Espace de noms d'URN d'identifiant unique universel (UUID)

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

Résumé
La présente spécification définit un espace de noms de nom de ressource universel (URN, *Uniform Resource Name*) pour les identifiants uniques universels (UUID, *Universally Unique IDentifier*) aussi appelés des identifiants uniques au monde (GUID, *Globally Unique IDentifier*). Un UUID fait 128 bits, et peut garantir son unicité dans l'espace et le temps. Les UUID ont été à l'origine utilisés dans le système de calcul du réseau Apollo et ensuite dans l'environnement de calcul réparti (DCE, *Distributed Computing Environment*) de la Fondation pour les logiciels ouverts (OSF, *Open Software Foundation*) et ensuite pour les plateformes Windows de Microsoft.

La présente spécification est dérivée de la spécification DCE avec l'aimable permission de OSF (qui s'appelle maintenant "The Open Group"). Les informations provenant de versions antérieures de la spécification DCE ont été incorporées dans le présent document.

Table des Matières

1. Introduction

La présente spécification définit un espace de noms de nom de ressource universel pour les identifiants uniques universels (UUID, *Universally Unique IDentifier*) aussi appelés des identifiants uniques au monde (GUID, *Globally Unique IDentifier*). Un UUID fait 128 bits, et n'exige pas de processus d'enregistrement central.

Les informations données ici sont destinées à être un guide concis pour ceux qui souhaitent mettre en œuvre des services utilisant des UUID comme URN. Rien dans le présent document ne devrait être interprété comme contredisant les standard DCE qui ont défini les UUID.

Une Recommandation de l'UIT-T qui est aussi une norme ISO/CEI [X.667] est dérivée des versions antérieures du présent document. Les deux ensembles de spécifications ont été alignés, et sont techniquement pleinement compatibles. De plus, une fonction d'enregistrement mondiale est assurée par le bureau de la normalisation des télécommunications de l'ITU-T ; pour les détails, voir < http://www.itu.int/ITU-T/asn1/uuid.html >.

2. Motivation

Une des principales raisons de l'utilisation des UUID est qu'aucune autorité centralisée n'est requise pour les administrer (bien qu'un format utilise les identifiants de nœud IEEE 802, et pas les autres). Par suite, la génération à la demande peut être complètement automatisée, et utilisée pour divers objets. L'algorithme de génération d'UUID décrit ici accepte de très hauts débits d'allocation jusqu'à 10 millions par seconde et par machine si nécessaire, de sorte qu'ils pourraient même être utilisés comme identifiants de transaction.

Les UUID sont de taille fixe (128 bits) ce qui est raisonnablement petit comparé aux autres solutions proposées. Ils se prêtent bien au tri, au rangement, et aux hachages de toutes sortes, à la mémorisation dans les bases de données, à la simple allocation, et à la programmation en général.

Comme les UUID sont uniques et persistants, ils font d'excellents noms de ressource universels. La capacité unique de générer un nouvel UUID sans processus d'enregistrement permet aux UUID d'être un des URN qui a le plus faible coût d'élaboration.

3. Gabarit d'enregistrement d'espace de noms

Identifiant d'espace de nom : UUID
Informations d'enregistrement :
Date d'enregistrement : 2003-10-01

Enregistrant déclaré de l'espace de noms : JTC 1/SC6 (ASN.1 Rapporteur Group)

Déclaration de la structure syntaxique : un UUID est un identifiant qui est unique à travers l'espace et le temps, par rapport à l'espace de tous les UUID. Comme un UUID est d'une taille fixe et contient un champ d'horodatage, il est possible aux valeurs de revenir à zéro (aux environs de l'an 3400, selon l'algorithme spécifique utilisé). Un UUID peut être utilisé pour de multiples objets, de l'étiquetage d'objets d'une durée de vie extrêmement courte, à l'identification fiable d'objets très persistants sur un réseau.

La représentation interne d'un UUID est une séquence spécifique de bits en mémoire, comme décrit à la Section 4. Pour représenter avec précision un UUID comme un URN, il est nécessaire de convertir la séquence binaire en représentation de chaîne.

Chaque champ est traité comme un entier et a sa valeur imprimée comme une chaîne de chiffres hexadécimaux remplie de zéros avec le chiffre de poids fort en premier. Les valeurs hexadécimale "a" à "f" sont sorties en caractères minuscules et sont insensibles à la casse en entrée.

La définition formelle de la représentation de chaîne d'UUID est fournie par l'ABNF [RFC2234] suivant :
UUID = time-low "-" time-mid "-" time-high-and-version "-" clock-seq-and-reserved clock-seq-low "-" node
time-low = 4hexOctet
time-mid = 2hexOctet
time-high-and-version = 2hexOctet
clock-seq-and-reserved = hexOctet
clock-seq-low = hexOctet
node = 6hexOctet
hexOctet = hexDigit hexDigit
hexDigit = "0" / "1" / "2" / "3" / "4" / "5" / "6" / "7" / "8" / "9" / "a" / "b" / "c" / "d" / "e" / "f" / "A" / "B" / "C" / "D" / "E" / "F"
Exemple de représentation de chaîne d'un UUID comme URN : urn:uuid:f81d4fae-7dec-11d0-a765-00a0c91e6bf6
Documentation annexe pertinente : [ARCHI] [DCE]
Considérations d'unicité d'identifiant : ce document spécifie trois algorithmes pour générer des UUID : le premier s'appuie sur les valeurs uniques des adresses MAC 802 pour garantir l'unicité, le second utilise des générateurs de nombres

pseudo aléatoires, et le troisième utilise un hachage cryptographique et des chaînes de texte fournies par l'application. Par suite, les UUID générés conformément à ce mécanisme seront uniques par rapport à tous les autres UUID qui ont été ou seront alloués.

Considérations de persistance d'identifiant : les UID sont par nature très difficiles à résoudre dans un sens global. Cela, couplé au fait que les UUID sont temporellement uniques au sein de leur contexte spatial, assure que les UUID vont rester aussi persistants que possible.

Processus d'allocation d'identifiant : générer un UUID n'exige pas qu'une autorité d'enregistrement soit contactée. Un algorithme exige une valeur unique sur l'espace pour chaque générateur. Cette valeur est normalement une adresse MAC IEEE 802, généralement déjà disponible sur les hôtes connectés au réseau. L'adresse peut être allouée à partir d'un bloc d'adresses obtenu de l'autorité d'enregistrement IEEE. Si une telle adresse n'est pas disponible, ou si des problèmes de confidentialité rendent son utilisation indésirable, le paragraphe 4.5 spécifie deux solutions de remplacement. Une autre approche est d'utiliser des UUID de version 3 ou de version 4 comme défini ci-dessous.

Processus de résolution d'identifiant : comme les UUID ne sont pas globalement résolubles, ceci n'est pas applicable.

Règles d'équivalence lexicale : on considère chaque champ de l'UUID comme étant un entier non signé comme montré dans le tableau du paragraphe 4.1.2. Pour comparer une paire d'UUID, on compare arithmétiquement les champs correspondants de chaque UUID dans l'ordre de signification et selon leur type de données. Deux UUID sont égaux si et seulement si tous les champs correspondants sont égaux.

Note de mise en œuvre : les comparaisons d'égalité peuvent être effectuées sur de nombreux systèmes en faisant la canonisation appropriée d'ordre des octets, et ensuite en traitant les deux UUID comme des entiers non signés de 128 bits.

Les UUID, comme définis dans le présent document, peuvent aussi être ordonnés lexicographiquement. Pour une paire d'UUID, le premier suit le second si le champ de plus fort poids dans lequel les UUID diffèrent est plus grand pour le premier UUID. Le second précède le premier si le champ de plus fort poids dans lequel les UUID diffèrent est plus grand pour le second UUID.

Conformité à la syntaxe d'URN : la représentation de chaîne d'un UUID est pleinement compatible avec la syntaxe d'URN. Lorsque on convertit d'une représentation en mode binaire en mémoire d'un UUID en URN, il faut faire attention à suivre strictement les questions d'ordre des octets mentionnées dans la section représentation de chaîne.

Mécanisme de validation : à part déterminer si la portion horodatage de l'UUID est dans le futur et donc non encore allouable, il n'y a pas de mécanisme pour déterminer si un UUID est "valide".

Portée : les UUID sont de portée mondiale.

4. Spécification

4.1 Format

Le format UUID est de 16 octets ; certains bits du champ de variante à huit octets spécifiée ci-dessous déterminent une structure plus fine.

4.1.1 Champ de variante

Le champ de variante détermine la disposition de l'UUID. C'est-à-dire, l'interprétation de tous les autres bits de l'UUID dépend du réglage des bits du champ de variante. À ce titre, il pourrait être plus précisément appelé un champ de type ; on conserve le terme original pour la compatibilité. Le champ de variante consiste en un nombre variable des bits de poids fort de l'octet 8 de l'UUID.

Le tableau suivant fait la liste des contenus du champ de variante, où la lettre "x" indique une valeur dont il n'y a pas à se soucier. *(Msb pour most significant bit, bit de poids fort)*

Msb0	Msb1	Msb2	Description
0	x	x	Réservé, pour la rétro compatibilité NCS.
1	0	x	Variante spécifiée dans le présent document.
1	1	0	Réservé, pour la rétro compatibilité Microsoft
1	1	1	Réservé pour future définition.

L'interopérabilité, sous toute ses formes, avec des variantes autres que celles définies ici n'est pas garantie, et n'est probablement pas un problème en pratique.

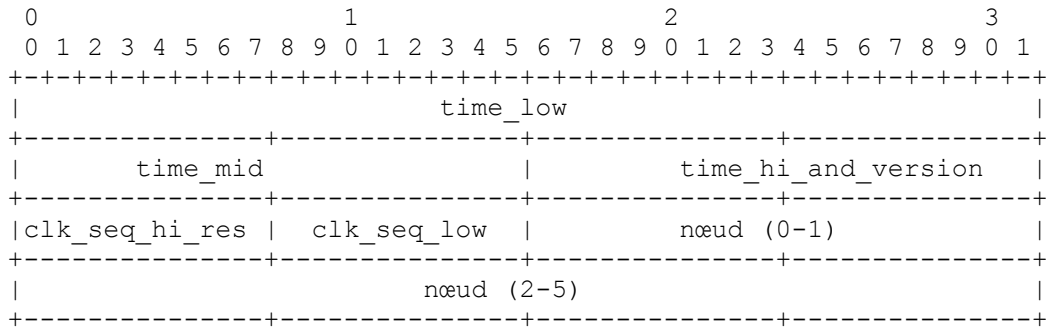
4.1.2 Présentation et ordre des octets

Pour minimiser la confusion sur l'allocation des bits au sein des octets, la définition d'enregistrement d'UUID est seulement en termes de champs qui sont des nombres entiers d'octets. Les champs sont présentés avec le poids fort en premier.

Champ	N ° de type de données	Octet	Note
time_low	entier non signé de 32 bits	0-3	Champ inférieur de l'horodatage
time_mid	entier non signé de 16 bits	4-5	Champ moyen de l'horodatage
time_hi_and_version	entier non signé de 16 bits	6-7	Champ supérieur de l'horodatage multiplexé avec le numéro de version
clock_seq_hi_and_reserved	entier non signé de 8 bits	8	Champ supérieur de la séquence d'horloge multiplexé avec la variante
clock_seq_low	entier non signé de 8 bits	9	Champ inférieur de la séquence d'horloge
node	entier non signé de 48 bits	10-15	Identifiant de nœud spatialement unique

En l'absence de spécification explicite de protocole d'application ou de présentation contraire, un UUID est codé comme objet de 128 bits, comme suit :

Les champs sont codés sur 16 octets, avec la taille et l'ordre des champs définis ci-dessus, et avec chaque champ codé avec l'octet de poids fort en premier (appelé ordre des octets du réseau). Noter que les noms des champs, en particulier pour les champs multiplexés, suivent la pratique historique.



4.1.3 Version

Le numéro de version est dans les 4 bits de poids fort de l'horodatage (bits 4 à 7 du champ time_hi_and_version).

Le tableau qui suit fait la liste des versions actuellement définies pour cette variante d'UUID.

Msb0	Msb1	Msb2	Msb3	Version	Description
0	0	0	1	1	Version fondée sur l'heure spécifiée dans ce document.
0	0	1	0	2	Version de sécurité DCE, avec UID POSIX incorporés.
0	0	1	1	3	Version fondée sur le nom spécifiée dans ce document avec hachage MD5.
0	1	0	0	4	Version générée de façon aléatoire ou pseudo aléatoire spécifiée dans ce document.
0	1	0	1	5	Version fondée sur le nom spécifiée dans ce document avec hachage SHA-1.

La version est plus précisément un sous type ; là encore, on conserve le terme pour la compatibilité.

4.1.4 Horodatage

L'horodatage est une valeur de 60 bits. Pour les UUID version 1, il est représenté par un temps universel coordonné (UTC) comme un compte d'intervalles de 100 nanosecondes depuis le 15 octobre 1582 à 00:00:00.00, (la date de la réforme grégorienne du calendrier chrétien).

Pour les systèmes qui ne disposent pas de l'UTC, mais ont l'heure locale, ils peuvent l'utiliser au lieu de l'UTC, pour autant qu'ils le fassent de façon cohérente sur tout le système. Cependant, ceci n'est pas recommandé car générer l'UTC à partir de l'heure locale demande seulement de connaître le décalage de la zone horaire.

Pour les UUID de version 3 ou 5, l'horodatage est une valeur de 60 bits construite à partir d'un nom, comme décrit au paragraphe 4.3.

Pour les UUID version 4, l'horodatage est une valeur de 60 bits générée de façon aléatoire ou pseudo aléatoire, comme décrit au paragraphe 4.4.

4.1.5 Séquence d'horloge

Pour les UUID version 1, la séquence d'horloge est utilisée pour aider à éviter les dupliqués qui pourraient survenir lorsque l'horloge est retardée ou si l'identifiant du nœud change.

Si l'horloge est retardée, ou peut avoir été retardée (par exemple, lorsque le système a été mis hors tension) et si le générateur de l'UUID ne peut être sûr qu'aucun UUID n'a été généré avec des horodatages supérieurs à la valeur à laquelle l'horloge a été réglée, la séquence d'horloge doit être changée. Si la valeur précédente de la séquence d'horloge est connue, elle peut être juste incrémentée ; autrement, elle devrait être réglé à une valeur aléatoire ou pseudo aléatoire de haute qualité.

De même, si l'identifiant de nœud change (par exemple, parce qu'une carte réseau a été déplacée d'une machine à une autre) régler la séquence d'horloge à un nombre aléatoire minimise la probabilité de dupliqué dû à de légères différences des réglages d'horloge sur les machines. Si la valeur de la séquence d'horloge associée à l'identifiant de nœud changé est connue, la séquence d'horloge pourrait alors être juste incrémentée, mais c'est peu probable.

La séquence d'horloge DOIT être à l'origine (c'est-à-dire, une fois dans la vie d'un système) initialisée à un nombre aléatoire pour minimiser la corrélation entre systèmes. Cela fournit une protection maximale contre les identifiants de nœud qui peuvent changer ou commuter rapidement d'un système à un autre. La valeur initiale NE DOIT PAS être corrélée à l'identifiant de nœud.

Pour les UUID version 3 ou 5, la séquence d'horloge est une valeur de 14 bits construite à partir d'un nom, comme décrit au paragraphe 4.3.

Pour les UUID version 4, la séquence d'horloge est une valeur aléatoire ou pseudo aléatoire de 14 bits, comme décrit au paragraphe 4.4.

4.1.6 Champ nœud

Pour les UUID version 1, le champ nœud consiste en une adresse MAC IEEE 802, généralement l'adresse de l'hôte. Pour les systèmes avec plusieurs adresses IEEE 802, toute adresse disponible peut être utilisée. Le plus petit octet adressé (octet numéro 10) contient le bit global/local et le bit envoi individuel/diffusion groupée, et c'est le premier octet de l'adresse transmise sur un LAN 802.3.

Pour les systèmes sans adresse IEEE, une valeur générée au hasard ou pseudo aléatoire peut être utilisée; voir le paragraphe 4.5. Le bit diffusion groupée doit être établi dans de telles adresses, afin qu'il n'y ait jamais de conflit avec les adresses obtenues de cartes réseau.

Pour les UUID version 3 ou 5, le champ nœud est une valeur de 48 bits construite à partir d'un nom comme décrit au paragraphe 4.3.

Pour les UUID version 4, le champ nœud est une valeur générée au hasard ou pseudo aléatoire de 48 bits, comme décrit au paragraphe 4.4.

4.1.7 UUID nil

L'UUID nil est une forme spéciale d'UUID qui est spécifiée comme ayant tous les 128 bits réglés à zéro.

4.2 Algorithmes pour créer un UUID fondé sur l'heure

Les divers aspects de l'algorithme de création d'UUID de version 1 sont discutés dans les paragraphes qui suivent.

4.2.1 Algorithme de base

L'algorithme qui suit est simple, correct, et inefficace :

- o Obtenir un verrou global à l'échelle du système.
- o À partir d'une mémorisation stable partagée à l'échelle du système (par exemple, un fichier) lire l'état du générateur d'UUID : les valeurs de l'horodatage, de séquence d'horloge, et d'identifiant de nœud utilisées pour générer le dernier

UUID.
- o Obtenir l'heure courante comme un compte sur 60 bits d'intervalles de 100 nanosecondes depuis le 15 octobre 1582 à 00:00:00.00.
- o Obtenir l'identifiant de nœud actuel.
- o Si l'état était indisponible (par exemple, non existent ou corrompu) ou si l'identifiant de nœud sauvegardé est différent de l'identifiant de nœud actuel, générer une valeur aléatoire de séquence d'horloge.
- o Si l'état était disponible, mais si l'horodatage sauvegardé est plus tard que l'horodatage actuel, incrémenter la valeur de séquence d'horloge.
- o Sauvegarder l'état (horodatage actuel, séquence d'horloge, et identifiant de nœud) dans la mémorisation stable.
- o Libérer le verrou global.
- o Formater un UUID à partir des valeurs d'horodatage actuel, de séquence d'horloge, et d'identifiant de nœud conformément aux étapes du paragraphe 4.2.2.

Si les UUID n'ont pas besoin d'être générés fréquemment, l'algorithme ci dessus peut être parfaitement adéquat. Pour des exigences de performances plus élevées, cependant, les problèmes posés par l'algorithme de base incluent :
- o Lire à chaque fois l'état à partir d'une mémorisation stable est inefficace.
- o La résolution de l'horloge système peut n'être pas 100 nanosecondes.
- o Écrire à chaque fois l'état sur une mémorisation stable est inefficace.
- o Partager l'état à travers les frontières de procès peut être inefficace.

Chacun de ces problèmes peut être traité de façon modulaire par des améliorations locales des fonctions qui lisent et écrivent l'état et lisent l'horloge. On traite tour à tour chacun d'eux dans les paragraphes qui suivent.

4.2.1.1 Lecture dans une mémorisation stable

L'état a seulement besoin d'être lu une fois à l'amorçage à partir d'une mémorisation stable, si il est lu dans une mémorisation volatile partagée à l'échelle du système (et mis à jour chaque fois que la mémorisation stable est mise à jour).

Si une mise en œuvre n'a pas de mémorisation stable disponible, elle peut toujours dire que les valeurs sont indisponibles. C'est la mise en œuvre la moins souhaitable parce que cela va augmenter la fréquence de création de nouveaux numéros de séquence d'horloge, ce qui augmente la probabilité de dupliqués.

Si l'identifiant de nœud ne peut jamais changer (par exemple, la carte réseau est inséparable du système) ou si tout changement réinitialise aussi la séquence d'horloge à une valeur aléatoire, au lieu de la conserver dans une mémorisation stable, l'identifiant de nœud actuel peut alors être retourné.

4.2.1.2 Résolution de l'horloge système

L'horodatage est généré à partir de l'heure système, dont la résolution peut être moindre que la résolution de l'horodatage d'UUID.

Si les UUID n'ont pas besoin d'être générés fréquemment, l'horodatage peut simplement être l'heure système multipliée par le nombre d'intervalles de 100 nanosecondes par intervalle de l'heure système.

Si un système submerge le générateur en demandant trop d'UUID dans un seul intervalle d'heure système, le service d'UUID DOIT retourner une erreur, ou mettre en pause le générateur d'UUID jusqu'à ce que l'horloge système reprenne le dessus.

Un horodatage à haute résolution peut être simulé en gardant un compte du nombre d'UUID qui ont été générés avec la même valeur de l'heure système, et en l'utilisant pour construire les bits de moindre poids de l'horodatage. Le compte va être entre zéro et le nombre d'intervalles de 100 nanosecondes par intervalle d'heure système.

Note : Si les processeurs submergent fréquemment la génération d'UUID, des identifiants de nœud supplémentaires peuvent être alloués au système, ce qui va permettre une allocation plus rapide en rendant plusieurs UUID potentiellement disponibles pour chaque valeur d'horodatage.

4.2.1.3 Écriture dans une mémorisation stable

L'état n'a pas toujours besoin d'être écrit dans une mémorisation stable chaque fois qu'un UUID est généré. L'horodatage dans la mémorisation stable peut être réglé périodiquement à une valeur supérieure à toute autre déjà utilisée dans un UUID. Tant que les UUID générés ont des horodatages inférieurs à cette valeur, et que la séquence d'horloge et l'identifiant de nœud restent inchangés, seule la copie volatile partagée de l'état a besoin d'être mise à jour. De plus, si la valeur de

l'horodatage dans la mémorisation stable est dans le futur de moins que le temps normal que prend le système pour réamorcer, une panne ne causera pas de réinitialisation de la séquence d'horloge.

4.2.1.4 Partage d'état à travers les processus

Si il est trop coûteux d'accéder à l'état partagé chaque fois qu'un UUID est généré, le générateur au niveau système peut alors être mis en œuvre pour allouer un bloc d'horodatage chaque fois qu'il est invoqué ; un générateur par processus peut allouer à partir de ce bloc jusqu'à ce qu'il soit épuisé.

4.2.2 Détails de génération

Les UUID de version 1 sont générés selon l'algorithme suivant :

- Déterminer les valeurs pour l'horodatage et la séquence d'horloge fondés sur UTC à utiliser dans l'UUID, comme décrit au paragraphe 4.2.1.
- Pour les besoins de cet algorithme, on considère l'horodatage comme un entier non signé de 60 bits et la séquence d'horloge comme un entier non signé de 14 bits. On numérote les bits à la suite dans un champ, en commençant par zéro pour le bit de moindre poids.
- On règle le champ time_low égal aux 32 bits de moindre poids (les bits de zéro à 31) de l'horodatage dans le même ordre de poids.
- On règle le champ time_mid égal aux bits de 32 à 47 provenant de l'horodatage dans le même ordre de poids.
- On règle les 12 bits de moindre poids (les bits de zéro à 11) du champ time_hi_and_version égaux aux bits 48 à 59 provenant de l'horodatage dans le même ordre de poids.
- On règle les quatre bits de plus fort poids (les bits 12 à 15) du champ time_hi_and_version au numéro de version de 4 bits correspondant à la version de l'UUID qu'on est en train de créer, comme montré dans le tableau ci-dessus.
- On règle le champ clock_seq_low aux huit bits de moindre poids (les bits zéro à 7) de la séquence d'horloge dans le même ordre de poids.
- On règle les 6 bits de moindre poids (les bits zéro à 5) du champ clock_seq_hi_and_reserved aux 6 bits de poids fort (les bits 8 à 13) de la séquence d'horloge dans le même ordre de poids.
- On règle les deux bits de poids fort (bits 6 et 7) du champ clock_seq_hi_and_reserved à zéro et un, respectivement.
- On règle le champ nœud de l'adresse IEEE de 48 bits dans le même ordre de poids que l'adresse.

4.3 Algorithme de création d'un UUID fondé sur le nom

L'UUID de version 3 ou 5 est destiné à générer des UUID à partir de "noms" qui sont tirés d'un "espace de noms", et sont uniques en son sein. Le concept de nom et d'espace de noms devrait être entendu au sens large, et non limité aux noms textuels. Par exemple, certains espaces de noms sont le système des noms de domaines, des URL, des identifiants d'objets ISO (OID), des noms distinctifs X.500 (DN), et des mots réservés dans un langage de programmation. Les mécanismes ou conventions utilisés pour allouer les noms et s'assurer de leur unicité au sein de leur espace de noms sortent du domaine d'application de la présente spécification.

Les exigences pour ces types d'UUID sont les suivantes :

- Les UUID générés à des moments différents à partir du même nom dans le même espace de noms DOIVENT être égaux.
- Les UUID générés à partir de deux noms différents dans le même espace de noms devraient être différents (avec une très forte probabilité).
- Les UUID générés à partir du même nom dans deux espaces de noms différents devraient être différents (avec une très forte probabilité).
- Si deux UUID qui ont été générés à partir de noms sont égaux, ils ont alors été générés à partir du même nom dans le même espace de noms (avec une très forte probabilité).

L'algorithme pour générer un UUID à partir d'un nom et d'un espace de noms est comme suit :

- Allouer un UUID à utiliser comme "identifiant d'espace de noms" pour tous les UUID générés à partir de noms dans cet espace de noms ; voir à l'Appendice C des valeurs prédéfinies.
- Choisir MD5 [RFC1321] ou SHA-1 [SHS] comme algorithme de hachage ; si la rétro compatibilité n'est pas un problème, SHA-1 est préféré.
- Convertir le nom en une séquence canonique d'octets (comme défini par les standard ou conventions de son espace de noms) ; mettre l'identifiant d'espace de noms dans l'ordre des octets du réseau.
- Calculer le hachage de l'identifiant d'espace de noms enchaîné avec le nom.
- Régler les octets de zéro à 3 du champ time_low aux octets zéro à 3 du hachage.
- Régler les octets zéro et un du champ time_mid aux octets 4 et 5 du hachage.
- Régler les octets zéro et un du champ time_hi_and_version aux octets 6 et 7 du hachage.

- Régler les quatre bits de poids fort (bits 12 à 15) du champ time_hi_and_version au numéro de version de 4 bits approprié du paragraphe Section 4.1.3.
- Régler le champ clock_seq_hi_and_reserved à l'octet 8 du hachage.
- Régler les deux bits de poids fort (bits 6 et 7) du champ clock_seq_hi_and_reserved à zéro et un, respectivement.
- Régler le champ clock_seq_low à l'octet 9 du hachage.
- Régler les octets zéro à cinq du champ nœud aux octets 10 à 15 du hachage.
- Convertir l'UUID résultant dans l'ordre local des octets.

4.4 Algorithmes pour la création d'un UUID à partir d'un nombre vraiment aléatoire ou pseudo aléatoire

L'UUID version 4 est destiné à générer des UUID à partir de nombres vraiment aléatoires ou pseudo aléatoires.

L'algorithme est le suivant :
- Régler des deux bits de poids fort (bits 6 et 7) du champ clock_seq_hi_and_reserved à zéro et un, respectivement.
- Régler les quatre bits de poids fort (bits 12 à 15) du champ time_hi_and_version au numéro de version à 4 bits du paragraphe 4.1.3.
- Régler tous les autres bits à des valeurs choisies au hasard (ou pseudo aléatoires).

Voir au paragraphe 4.5 une discussion sur les nombres aléatoires.

4.5 Identifiants de nœuds qui n'identifient pas l'hôte

Ce paragraphe décrit comment générer un UUID de version 1 si une adresse IEEE 802 n'est pas disponible, ou si son utilisation n'est pas désirée.

Une approche est de contacter l'IEEE et d'obtenir un bloc d'adresses séparé. Au moment de cette rédaction, l'application pouvait être trouvée à < http://standards.ieee.org/regauth/oui/pilot-ind.html >, et le coût était de 550 $ US.

Une meilleure solution est d'obtenir un numéro aléatoire de qualité cryptographique de 47 bits et de l'utiliser comme les 47 bits de moindre poids de l'identifiant de nœud, avec le bit de moindre poids du premier octet de l'identifiant de nœud réglé à un. Ce bit est le bit envoi individuel/diffusion groupée, qui ne sera jamais établi dans les adresses IEEE 802 obtenues des cartes réseau. Donc, il ne peut jamais y avoir de conflit entre les UUID générés par des machines avec et sans carte réseau. (On se rappelle que la spécification IEEE 802 parle d'ordre de transmission, ce qui est à l'opposé de la représentation en mémoire qui est présentée dans ce document.)

Pour la compatibilité avec les spécifications antérieures, on notera que le présent document utilise le bit envoi individuel/diffusion groupée, au lieu du bit discutablement plus correct local/global.

Un avis sur la génération de nombres aléatoires de qualité cryptographique se trouve dans la [RFC4086].

De plus, des éléments comme le nom de l'ordinateur et le nom du système d'exploitation, bien qu'à proprement parler non strictement aléatoires, vont aider à différencier les résultats de ceux obtenus par d'autres systèmes.

L'algorithme exact pour générer un identifiant de nœud en utilisant ces données est spécifique du système, parce que les données disponibles et les fonctions pour les obtenir sont souvent très spécifiques des systèmes. Ube approche générique est cependant d'accumuler autant de sources que possible dans une mémoire tampon, d'utiliser un résumé de message comme MD5 [RFC1321] ou SHA-1 [SHS], de prendre six octets arbitraires d'une valeur de hachage, et de régler le bit de diffusion groupée comme décrit ci-dessus.

5. Considérations de communauté

L'utilisation des UUID est extrêmement présente en informatique. Cela englobe le cœur de l'infrastructure d'identifiant pour de nombreux systèmes d'exploitation (Microsoft Windows) et applications (le navigateur Mozilla) et dans de nombreux cas, devient exposé sur la Toile de nombreuses façons non standard.

La présente spécification tente de normaliser cette pratique de façon aussi ouverte que possible et pour le bénéfice de l'Internet tout entier.

6. Considérations sur la sécurité

On ne doit pas supposer que les UUID sont difficiles à deviner ; ils ne devraient pas être utilisés comme moyen de sécurité (comme identifiants dont la simple possession accorde l'accès) par exemple. Une source de nombres aléatoires prévisibles va exacerber la situation.

On ne doit pas supposer qu'il est facile de déterminer si un UUID a été légèrement transposé afin de rediriger une référence sur un autre objet. Les humains n'ont pas la capacité de vérifier facilement l'intégrité d'un UUID par un simple coup d'œil.

Les applications réparties qui génèrent des UUID chez divers hôtes doivent vouloir s'appuyer sur la source de nombres aléatoires chez tous les hôtes. Si ce n'est pas faisable, la variante d'espace de noms devrait être utilisée.

7. Remerciements

Le présent document s'appuie fortement sur la spécification DCE OSF pour les UUID. Ted Ts'o a fourni d'utiles commentaires, en particulier sur la section d'ordre des octets qui est presque entièrement copiée de la proposition qu'il a fournie (mais toutes les erreurs dans cette section sont cependant de notre responsabilité).

Nous remercions aussi de leur relecture attentive Ralf S. Engelschall, John Larmouth, et Paul Thorpe. La collaboration du professeur Larmouth a aussi été précieuse pour la coordination avec l'ISO/CEI.

8. Références normatives

[ARCHI] Zahn, L., Dineen, T., and P. Leach, "Network Computing Architecture", ISBN 0-13-611674-4, janvier 1990.

[DCE] Open Group CAE Specification C309, "DCE: Remote Procedure Call", ISBN 1-85912-041-5, août 1994.

[RFC1321] R. Rivest, "Algorithme de résumé de message MD5", avril 1992. *(Information)*

[RFC2141] R. Moats, "Syntaxe des URN", mai 1997. *(Obsolète, voir* RFC8141)

[RFC2234] D. Crocker et P. Overell, "BNF augmenté pour les spécifications de syntaxe : ABNF", novembre 1997. (*Obsolète, voir* RFC5234)

[RFC4086] D. Eastlake 3rd, J. Schiller, S. Crocker, "Exigences d'aléa pour la sécurité", juin 2005. (*Remplace* RFC1750) (BCP0106)

[SHS] National Institute of Standards and Technology, "Secure Hash Standard", FIPS PUB 180-1, avril 1995, <http://www.itl.nist.gov/fipspubs/fip180-1.htm>.

[X.667] ISO/CEI 9834-8:2004, "Technologie de l'information - Procédures pour le fonctionnement des autorités d'enregistrement OSI : génération et enregistrement des identifiants uniques au monde (UUID) et leur utilisation comme composants d'identifiants d'objets ASN.1" Recommandation UIT-T X.667, 2004.

Appendice A Échantillon de mise en œuvre

Cette mise en œuvre consiste en 5 fichiers : uuid.h, uuid.c, sysdep.h, sysdep.c et utest.c. Les fichiers uuid.* files sont la mise en œuvre indépendante du système des algorithmes de génération d'UUID décrits ci-dessus avec toutes les optimisations décrites ci-dessus sauf le partage d'état efficace à travers les processus inclus. Le code a été essayé sur Linux (Red Hat 4.0) avec GCC (2.7.2), et Windows NT 4.0 avec VC++ 5.0. Le code suppose la prise en charge d'entiers de 64 bits, ce qui le rend beaucoup plus clair.

Tous les fichiers sources suivants devraient inclure la notice de droits de reproduction suivante :

copyrt.h

/*

uuid.h

```
#include "copyrt.h"
#undef uuid_t
typedef struct {
   unsigned32  time_low;
   unsigned16  time_mid;
   unsigned16  time_hi_and_version;
   unsigned8   clock_seq_hi_and_reserved;
   unsigned8   clock_seq_low;
   byte        node[RFC2141];
} uuid_t;

/* uuid_create – génère un UUID */
int uuid_create(uuid_t * uuid);

/* uuid_create_md5_from_name – crée un UUID version 3 (MD5) utilisant un "nom" provenant d'un "espace de noms" */
void uuid_create_md5_from_name(
   uuid_t *uuid,        /* UUID résultant */
   uuid_t nsid,         /* UUID de l'espace de noms */
   void *name,          /* nom à partir duquel on génère un UUID */
   int namelen          /* longueur du nom */
);

/* uuid_create_sha1_from_name – crée un UUID version 5 (SHA-1) utilisant un "nom" provenant d'un "espace de noms"
*/
void uuid_create_sha1_from_name(
   uuid_t *uuid,        /* UUID résultant */
   uuid_t nsid,         /* UUID de l'espace de noms */
   void *name,          /* nom à partir duquel on génère un UUID */
   int namelen          /* longueur du nom */
);

/* uuid_compare --  Compare deux UUID "lexicalement" et retour :
       -1   u1 est lexicalement avant u2
        0   u1 est égal à u2
        1   u1 est lexicalement après u2
Noter que l'ordre lexical n'est pas l'ordre temporel !
*/
int uuid_compare(uuid_t *u1, uuid_t *u2);
```

uuid.c

```
#include "copyrt.h"
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include "sysdep.h"
#include "uuid.h"
```

```
/* diverses déclarations préalables */
static int read_state(unsigned16 *clockseq, uuid_time_t *timestamp, uuid_node_t *node);
static void write_state(unsigned16 clockseq, uuid_time_t timestamp, uuid_node_t node);
static void format_uuid_v1(uuid_t *uuid, unsigned16 clockseq, uuid_time_t timestamp, uuid_node_t node);
static void format_uuid_v3or5(uuid_t *uuid, unsigned char hash[16], int v);
static void get_current_time(uuid_time_t *timestamp);
static unsigned16 true_random(void);

/* uuid_create – génère un UUID */
int uuid_create(uuid_t *uuid)
{
    uuid_time_t timestamp, last_time;
    unsigned16 clockseq;
    uuid_node_t node;
    uuid_node_t last_node;
    int f;

/* acquière un verrou de niveau système de sorte qu'on est entre nous */
    LOCK;
/* obtient l'heure, l'identifiant de nœud, l'état sauvegardé d'une mémorisation non volatile */
    get_current_time(&timestamp);
    get_ieee_node_identifier(&node);
    f = read_state(&clockseq, &last_time, &last_node);

/* si on n'est pas dans l'état NV, ou si l'horloge a été retardée, ou si l'identifiant de nœud a changé (par exemple, une
nouvelle carte réseau) changer clockseq (la séquence d'horloge) */
    if (!f || memcmp(&node, &last_node, sizeof node))
       clockseq = true_random();
    else if (timestamp < last_time)
       clockseq++;

/* sauvegarder l'état pour la prochaine fois */
    write_state(clockseq, timestamp, node);

    UNLOCK;

/* Champs de substance dans l'UUID */
    format_uuid_v1(uuid, clockseq, horodatage, node);
    return 1;
}

/* format_uuid_v1 – construit un UUID à partir de l'horodatage, de la séquence d'horloge, et l'identifiant de nœud */
void format_uuid_v1(uuid_t* uuid, unsigned16 clock_seq, uuid_time_t timestamp, uuid_node_t node)
{
 /* Construit un UUID version 1 avec les informations rassemblées plus quelques constantes. */
   uuid->time_low = (unsigned long)(timestamp & 0xFFFFFFFF);
   uuid->time_mid = (unsigned short)((timestamp >> 32) & 0xFFFF);
   uuid->time_hi_and_version = (unsigned short)((timestamp >> 48) & 0x0FFF);
   uuid->time_hi_and_version |= (1 << 12);
   uuid->clock_seq_low = clock_seq & 0xFF;
   uuid->clock_seq_hi_and_reserved = (clock_seq & 0x3F00) >> 8;
   uuid->clock_seq_hi_and_reserved |= 0x80;
   memcpy(&uuid->node, &node, sizeof uuid->node);
}

/* type de données pour l'état persistant de générateur d'UUID */
typedef struct {
   uuid_time_t  ts;         /* horodatage */
   uuid_node_t  node;    /*  identifiant de nœud sauvegardé */
   unsigned16   cs;         /* séquence d'horloge sauvegardée */
} uuid_state;
```

```
static uuid_state st;

/* read_state – lit l'état du générateur d'UUID dans la mémorisation non volatile */
int read_state(unsigned16 *clockseq, uuid_time_t *timestamp, uuid_node_t *node)
{
   static int inited = 0;
   FILE *fp;

/* il est seulement nécessaire de lire l'état un fois par amorçage */
   if (!inited) {
      fp = fopen("state", "rb");
      if (fp == NULL)
         return 0;
      fread(&st, sizeof st, 1, fp);
      fclose(fp);
      inited = 1;
   }
   *clockseq = st.cs;
   *timestamp = st.ts;
   *node = st.node;
   return 1;
}

/* write_state – sauvegarde l'état de générateur UUID de nouveau dans la mémorisation non volatile */
void write_state(unsigned16 clockseq, uuid_time_t timestamp, uuid_node_t node)
{
   static int inited = 0;
   static uuid_time_t next_save;
   FILE* fp;

   if (!inited) {
      next_save = timestamp;
      inited = 1;
   }

/* toujours sauvegarder l'état sur un état partagé volatile */
   st.cs = clockseq;
   st.ts = timestamp;
   st.node = node;
   if (timestamp >= next_save) {
      fp = fopen("state", "wb");
      fwrite(&st, sizeof st, 1, fp);
      fclose(fp);
      /* programme la prochaine sauvegarde dans 10 secondes à partir de maintenant */
      next_save = timestamp + (10 * 10 * 1000 * 1000);
   }
}

/* get-current_time – obtient l'heure comme des tics de 100 ns de 60 bits depuis l'époque des UUID. Compense le fait que
la résolution d'horloge réelle est moins que 100 ns. */
void get_current_time(uuid_time_t *timestamp)
{
   static int inited = 0;
   static uuid_time_t time_last;
   static unsigned16 uuids_this_tick;
   uuid_time_t time_now;

   if (!inited) {
      get_system_time(&time_now);
      uuids_this_tick = UUIDS_PER_TICK;
      inited = 1;
   }
```

```
   for ( ; ; ) {
      get_system_time(&time_now);

/* si la lecture d'horloge a changé depuis que le dernier UUID a été généré, */
      if (time_last != time_now) {
/* rétablir le compte d'uuid générés avec cette lecture d'horloge */
         uuids_this_tick = 0;
         time_last = time_now;
         break;
      }
      if (uuids_this_tick < UUIDS_PER_TICK) {
         uuids_this_tick++;
         break;
      }

/* va trop vite pour notre hologe ; spin */
   }
/* ajoute le compte d'uuid aux bits de moindre poids de la lecture d'horloge */
   *timestamp = time_now + uuids_this_tick;
}

/* true_random -- génère un nombre aléatoire de qualité cryptographique. **Cet exemple ne le fait pas.** */
static unsigned16 true_random(void)
{
   static int inited = 0;
   uuid_time_t time_now;

   if (!inited) {
      get_system_time(&time_now);
      time_now = time_now / UUIDS_PER_TICK;
      srand((unsigned int)
            (((time_now >> 32) ^ time_now) & 0xffffffff));
      inited = 1;
   }

   return rand();
}

/* uuid_create_md5_from_name – crée une version 3 (MD5) d'UUID utilisant un "nom" provenant d'un "espace de noms"
*/
void uuid_create_md5_from_name(uuid_t *uuid, uuid_t nsid, void *name, int namelen)
{
   MD5_CTX c;
   unsigned char hash[16];
   uuid_t net_nsid;

/* Met l'identifiant d'espace de noms dans l'ordre des octets du réseau de façon qu'il soit haché sans considération de si
l'ordre de la machine est gos boutien ou non */
   net_nsid = nsid;
   net_nsid.time_low = htonl(net_nsid.time_low);
   net_nsid.time_mid = htons(net_nsid.time_mid);
   net_nsid.time_hi_and_version = htons(net_nsid.time_hi_and_version);

   MD5Init(&c);
   MD5Update(&c, &net_nsid, sizeof net_nsid);
   MD5Update(&c, name, namelen);
   MD5Final(hash, &c);

/* Le hachage est à ce point dans l'ordre des octets du réseau */
   format_uuid_v3or5(uuid, hash, 3);
}

void uuid_create_sha1_from_name(uuid_t *uuid, uuid_t nsid, void *name, int namelen)
```

```
{
   SHA_CTX c;
   unsigned char hash[20];
   uuid_t net_nsid;

/* Met l'identifiant d'espace de noms dans l'ordre des octets du réseau de façon qu'il soit haché sans considération de si
l'ordre de la machine est gos boutien ou non */
   net_nsid = nsid;
   net_nsid.time_low = htonl(net_nsid.time_low);
   net_nsid.time_mid = htons(net_nsid.time_mid);
   net_nsid.time_hi_and_version = htons(net_nsid.time_hi_and_version);

   SHA1_Init(&c);
   SHA1_Update(&c, &net_nsid, sizeof net_nsid);
   SHA1_Update(&c, name, namelen);
   SHA1_Final(hash, &c);

/* Le hachage est à ce point dans l'ordre des octets du réseau */
   format_uuid_v3or5(uuid, hash, 5);
}

/* format_uuid_v3or5 – construit un UUID à partir d'un nombre (pseudo)aléatoire de 128 bits */
void format_uuid_v3or5(uuid_t *uuid, unsigned char hash[16], int v)
{
   /* convertit l'UUID en ordre des octets local */
   memcpy(uuid, hash, sizeof *uuid);
   uuid->time_low = ntohl(uuid->time_low);
   uuid->time_mid = ntohs(uuid->time_mid);
   uuid->time_hi_and_version = ntohs(uuid->time_hi_and_version);

   /* introduit les bits variante et version */
   uuid->time_hi_and_version &= 0x0FFF;
   uuid->time_hi_and_version |= (v << 12);
   uuid->clock_seq_hi_and_reserved &= 0x3F;
   uuid->clock_seq_hi_and_reserved |= 0x80;
}

/* uuid_compare --  Compare deux IID "lexicalement" et retour */
#define CHECK(f1, f2) if (f1 != f2) return f1 < f2 ? -1 : 1;
int uuid_compare(uuid_t *u1, uuid_t *u2)
{
   int i;

   CHECK(u1->time_low, u2->time_low);
   CHECK(u1->time_mid, u2->time_mid);
   CHECK(u1->time_hi_and_version, u2->time_hi_and_version);
   CHECK(u1->clock_seq_hi_and_reserved, u2->clock_seq_hi_and_reserved);
   CHECK(u1->clock_seq_low, u2->clock_seq_low)
   for (i = 0; i < 6; i++) {
      if (u1->node[i] < u2->node[i])
         return -1;
      if (u1->node[i] > u2->node[i])
         return 1;
   }
   return 0;
}
#undef CHECK


sysdep.h

#include "copyrt.h"
/* retirer ce qui suit définit si on fonctionne avec WIN32 */
```

```
#define WININC 0

#ifdef WININC
#include <windows.h>
#else
#include <sys/types.h>
#include <sys/time.h>
#include <sys/sysinfo.h>
#endif

#include "global.h"
/* Changer pour pointer sur le lieu de résidence du hachage MD5. La RFC 1321 a un exemple de mise en œuvre */
#include "md5.h"

/* Régler ce qui suit au nombre de tics de 100 ns de la résolution réelle de l'horloge système */
#define UUIDS_PER_TICK 1024

/* Régler ce qui suit à un appel pour obtenir et libérer un verrou global */
#define LOCK
#define UNLOCK

typedef unsigned long   unsigned32;
typedef unsigned short  unsigned16;
typedef unsigned char   unsigned8;
typedef unsigned char   byte;

/* Régler cela à ce que l'ordinateur utilise pour le type de données à 64 bits */
#ifdef WININC
#define unsigned64_t unsigned __int64
#define I64(C) C
#else
#define unsigned64_t unsigned long long
#define I64(C) C##LL
#endif

typedef unsigned64_t uuid_time_t;
typedef struct {
   char nodeID[RFC2141];
} uuid_node_t;

void get_ieee_node_identifier(uuid_node_t *node);
void get_system_time(uuid_time_t *uuid_time);
void get_random_info(char seed[16]);
```

sysdep.c

```
#include "copyrt.h"
#include <stdio.h>
#include "sysdep.h"

/* Appel dépendant du système pour obtenir un identifiant de nœud IEEE. Cet échantillon de mise en œuvre génère un
identifiant de nœud aléatoire. */
void get_ieee_node_identifier(uuid_node_t *node)
{
   static inited = 0;
   static uuid_node_t saved_node;
   char seed[16];
   FILE *fp;

   if (!inited) {
      fp = fopen("nodeid", "rb");
      if (fp) {
         fread(&saved_node, sizeof saved_node, 1, fp);
```

```
            fclose(fp);
        }
        else {
            get_random_info(seed);
            seed[0] |= 0x01;
            memcpy(&saved_node, seed, sizeof saved_node);
            fp = fopen("nodeid", "wb");
            if (fp) {
                fwrite(&saved_node, sizeof saved_node, 1, fp);
                fclose(fp);
            }
        }

        inited = 1;
    }

    *node = saved_node;
}

/* Appel dépendant du système pour obtenir l'heure système courante. Retournée comme tics de 100 ns depuis l'époque
d'UUID, mais la résolution peut être inférieure à 100 ns. */
#ifdef _WINDOWS_

void get_system_time(uuid_time_t *uuid_time)
{
    ULARGE_INTEGER time;

/* NT conserve l'heure en format FILETIME qui est des tics de 100 ns depuis le 1er janvier 1601. Les UUID utilisent
l'heurre en tics de 100 ns depuis le 15 octobre 1582. La différence est de 17 jours en octobre + 30 (novembre) + 31
(décembre) + 18 ans et  5 jours d'ajustement. */
    GetSystemTimeAsFileTime((FILETIME *)&time);
    time.QuadPart +=

          (unsigned __int64) (1000*1000*10)       // secondes
        * (unsigned __int64) (60 * 60 * 24)       // jours
        * (unsigned __int64) (17+30+31+365*18+5); // différence de jours
    *uuid_time = time.QuadPart;
}

/* Échantillon de code, à ne pas utiliser dans la réalité, voir la [RFC4086] */
void get_random_info(char seed[16])
{
    MD5_CTX c;
    struct {
        MEMORYSTATUS m;
        SYSTEM_INFO s;
        FILETIME t;
        LARGE_INTEGER pc;
        DWORD tc;
        DWORD l;
        char hostname[MAX_COMPUTERNAME_LENGTH + 1];
    } r;

    MD5Init(&c);
    GlobalMemoryStatus(&r.m);
    GetSystemInfo(&r.s);
    GetSystemTimeAsFileTime(&r.t);
    QueryPerformanceCounter(&r.pc);
    r.tc = GetTickCount();

    r.l = MAX_COMPUTERNAME_LENGTH + 1;
    GetComputerName(r.hostname, &r.l);
    MD5Update(&c, &r, sizeof r);
```

```
    MD5Final(seed, &c);
}

#else

void get_system_time(uuid_time_t *uuid_time)
{
    struct timeval tp;

    gettimeofday(&tp, (struct timezone *)0);

/* Décalage entre l'heure formatée d'UUID et l'heure formatée en Unix. La base des temps UTC d'UUID est le
15 octobre 1582. La base des temps Unix est le 1er janvier 1970.*/
    *uuid_time = ((unsigned64)tp.tv_sec * 10000000)
        + ((unsigned64)tp.tv_usec * 10)
        + I64(0x01B21DD213814000);
}

/* Échantillon de code, à ne pas utiliser dans la réalité, voir la [RFC4086] */
void get_random_info(char seed[16])
{
    MD5_CTX c;
    struct {
        struct sysinfo s;
        struct timeval t;
        char hostname[257];
    } r;

    MD5Init(&c);
    sysinfo(&r.s);
    gettimeofday(&r.t, (struct timezone *)0);
    gethostname(r.hostname, 256);
    MD5Update(&c, &r, sizeof r);
    MD5Final(seed, &c);
}

#endif
```

utest.c

```
#include "copyrt.h"
#include "sysdep.h"
#include <stdio.h>
#include "uuid.h"

uuid_t NameSpace_DNS = { /* 6ba7b810-9dad-11d1-80b4-00c04fd430c8 */
    0x6ba7b810,
    0x9dad,
    0x11d1,
    0x80, 0xb4, 0x00, 0xc0, 0x4f, 0xd4, 0x30, 0xc8
};

/* puid – imprime un UUID */
void puid(uuid_t u)
{
    int i;

    printf("%8.8x-%4.4x-%4.4x-%2.2x%2.2x-", u.time_low, u.time_mid,
    u.time_hi_and_version, u.clock_seq_hi_and_reserved,
    u.clock_seq_low);
    for (i = 0; i < 6; i++)
        printf("%2.2x", u.node[i]);
    printf("\n");
```

```
}

/* Simple pilote pour générateur d'UUID */
void main(int argc, char **argv)
{
   uuid_t u;
   int f;

   uuid_create(&u);
   printf("uuid_create(): "); puid(u);

   f = uuid_compare(&u, &u);
   printf("uuid_compare(u,u): %d\n", f);                    /* devrait être 0 */
   f = uuid_compare(&u, &NameSpace_DNS);
   printf("uuid_compare(u, NameSpace_DNS): %d\n", f); /* s.b. 1 */
   f = uuid_compare(&NameSpace_DNS, &u);
   printf("uuid_compare(NameSpace_DNS, u): %d\n", f); /* s.b. -1 */
   uuid_create_md5_from_name(&u, NameSpace_DNS, "www.widgets.com", 15);
   printf("uuid_create_md5_from_name(): "); puid(u);
}
```

Appendice B Échantillon de sortie de utest

```
uuid_create(): 7d444840-9dc0-11d1-b245-5ffdce74fad2
uuid_compare(u,u): 0
uuid_compare(u, NameSpace_DNS): 1
uuid_compare(NameSpace_DNS, u): -1
uuid_create_md5_from_name(): e902893a-9d22-3c7e-a7b8-d6e313b71d9f
```

Appendice C Quelques identifiants d'espace de noms

Cet appendice donne les identifiants d'espace de noms pour quelques espaces de noms potentiellement intéressants, comme des structures C initialisées et dans la représentation de chaîne définie ci-dessus.

```
/* La chaîne de noms est un nom de domaine pleinement qualifié */
uuid_t NameSpace_DNS = { /* 6ba7b810-9dad-11d1-80b4-00c04fd430c8 */
   0x6ba7b810,
   0x9dad,
   0x11d1,
   0x80, 0xb4, 0x00, 0xc0, 0x4f, 0xd4, 0x30, 0xc8
};

/* La chaîne de noms est un URL */
uuid_t NameSpace_URL = { /* 6ba7b811-9dad-11d1-80b4-00c04fd430c8 */
   0x6ba7b811,
   0x9dad,
   0x11d1,
   0x80, 0xb4, 0x00, 0xc0, 0x4f, 0xd4, 0x30, 0xc8
};

/* La chaîne de noms est un OID ISO */
uuid_t NameSpace_OID = { /* 6ba7b812-9dad-11d1-80b4-00c04fd430c8 */
   0x6ba7b812,
   0x9dad,
   0x11d1,
   0x80, 0xb4, 0x00, 0xc0, 0x4f, 0xd4, 0x30, 0xc8
};

/* La chaîne de noms est un nom distinctif X.500 (en DER ou en format de sortie texte) */
uuid_t NameSpace_X500 = { /* 6ba7b814-9dad-11d1-80b4-00c04fd430c8 */
```

```
    0x6ba7b814,
    0x9dad,
    0x11d1,
    0x80, 0xb4, 0x00, 0xc0, 0x4f, 0xd4, 0x30, 0xc8
};
```

Adresse des auteurs

Paul J. Leach
Microsoft
1 Microsoft Way
Redmond, WA 98052
US
téléphone : +1 425-882-8080
mél : paulle@microsoft.com

Michael Mealling
Refactored Networks, LLC
1635 Old Hwy 41
Suite 112, Box 138
Kennesaw, GA 30152
US
téléphone : +1-678-581-9656
mél : michael@refactored-networks.com
URI : http://www.refactored-networks.com

Rich Salz
DataPower Technology, Inc.
1 Alewife Center
Cambridge, MA 02142
US
téléphone : +1 617-864-0455
mél : rsalz@datapower.com
URI : http://www.datapower.com

Déclaration complète de droits de reproduction

Remerciement
Le financement de la fonction d'édition des RFC est fourni par la Administrative Support Activity (IASA) de l'IETF