RFC 9562 : Universally Unique IDentifiers (UUIDs)

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 12 mai 2024

Date de publication du RFC : Mai 2024

https://www.bortzmeyer.org/9562.html

Ce RFC normalise les UUID, une famille d'identificateurs uniques, obtenus sans registre central. Il remplace l'ancienne norme, le RFC 4122 [1], avec pas mal de nouveautés et un RFC complètement refait.

Les UUID, également connus autrefois sous le nom de GUID ("*Globally Unique IDentifiers*"), sont issus à l'origine du système des Apollo, adopté ensuite dans la plate-forme DCE. Les UUID ont une taille fixe, 128 bits, et sont obtenus localement, par exemple à partir d'un autre identificateur unique comme un nom de domaine, ou bien en tirant au hasard, la grande taille de leur espace de nommage faisant que les collisions sont très improbables (section 6.7 du RFC). Ce RFC reprend la spécification (bien oubliée aujourd'hui) de DCE de l'Open Group (ex-OSF) et ajoute la définition d'un espace de noms pour des URN (sections 4 et 7). (Il existe aussi une norme ITU sur les UUID et un registre des UUID <https://www.itu.int/en/ITU-T/asn1/Pages/UUID/uuids.aspx>, pour ceux qui y tiennent.)

Les UUID peuvent donc convenir pour identifier une entité sur le réseau, par exemple une machine mais aussi, vu leur nombre, comme identificateur unique pour des transactions (ce qui était un de leurs usages dans DCE). En revanche, ils ne sont pas résolvables, contrairement aux noms de domaine. Mais ils sont présents dans beaucoup de logiciels (Windows, par exemple, les utilise intensivement). On les utilise comme clé primaire dans les bases de données, comme identificateur de transaction, comme nom de machine, etc.

Les UUID peuvent se représenter sous forme d'un nombre binaire de 128 bits (la section 5 du RFC décrit les différents champs qui peuvent apparaitre) ou bien sous forme texte. Sur Unix, on peut fabriquer un UUID avec la commande `uuidgen`, qui affiche la représentation texte standard que normalise notre RFC (section 4) :

1. Pour voir le RFC de numéro NNN, https://www.ietf.org/rfc/rfcNNN.txt, par exemple https://www.ietf.org/rfc/rfc4122.txt

```
% uuidgen
317e8ed3-1428-4ef1-9dce-505ffbcba11a

% uuidgen
ec8638fd-c93d-4c6f-9826-f3c71436443a
```

Comme vous avez vu, aucune connexion à un registre central n'est nécessaire, aucune coordination n'est utile. C'est un élément essentiel du cahier des charges des UUID.

Sur Linux, vous pouvez aussi simplement faire `cat /proc/sys/kernel/random/uuid`. Sur une machine FreeBSD, un UUID de la machine est automatiquement généré (par le script `/etc/rc.d/hostid`) et stocké dans le fichier `/etc/hostid`.

Pour l'affichage sous forme d'URN (RFC 8141), on ajoute juste l'espace `uuid` par exemple `urn:uuid:ec8638fd-c`
Il a été ajouté au registre IANA des espaces de noms des URN <`https://www.iana.org/assignments/urn-namespaces/urn-namespaces.xml#urn-namespaces-1`>.

La section 4 du RFC détaille le format de l'UUID. En dépit des apparences, l'UUID n'est pas plat, il a une structure, mais il est très déconseillé aux applications de l'interpréter (section 6.12). Un des champs les plus importants est le champ Version (qui devrait plutôt s'appeler Type) car il existe plusieurs types d'UUID :

- UUID version 1, fondé sur le temps (notez tout de suite que les versions 6 et 7 sont recommandées à sa place).
- UUID version 2, réservé, non utilisé.
- UUID version 3, fondé sur un espace de noms (par exemple le DNS) et un nom qui est condensé en MD5 (la version 5 est préférable).
- UUID version 4, aléatoire.
- UUID version 5, fondé sur un espace de noms et un nom qui est condensé en SHA-1.
- UUID version 6, fondé sur le temps, avec les mêmes champs que la version 1, mais avec un format différent.
- UUID version 7, fondé sur le temps, avec une autre définition, et des champs différents.
- UUID version 8, pour les usages expérimentaux, si vous voulez jouer localement avec un nouveau mécanisme de génération.

Ces différents types / versions figurent dans un registre IANA <`https://www.iana.org/assignments/uuid/uuid.xml#uuid-subtypes`>. Ce registre ne peut être modifié que par une action de normalisation (cf. RFC 8126).

`uuidgen`, vu plus haut, peut générer des UUID de version 1 option `-t`, de version 3 (`-m`), de version 4 (c'est son comportement par défaut, mais on peut utiliser l'option `-r` si on veut être explicite) ou de version 5 (`-s`). Ici, on voit les UUID fondés sur une estampille temporelle (version 1) augmenter petit à petit :

```
% uuidgen -t
42ff1626-0fc7-11ef-8162-49e9505fb2f3

% uuidgen -t
4361fae8-0fc7-11ef-8162-49e9505fb2f3

% uuidgen -t
45381d02-0fc7-11ef-8162-49e9505fb2f3
```

Ici, dans le cas d'un UUID fondé sur un nom (version 3), l'UUID est stable (essayez chez vous, vous devriez obtenir le même résultat que moi), une propriété importante des UUID de version 3 et 5 :

```
% uuidgen -m -n @dns -N foobar.example
8796bf1a-793c-3c44-9ec5-a572635cd3d4

% uuidgen -m -n @dns -N foobar.example
8796bf1a-793c-3c44-9ec5-a572635cd3d4
```

Les espaces de noms sont enregistrés dans un registre IANA <https://www.iana.org/assignments/uuid/uuid.xml#uuid-namespace-ids>, d'autres peuvent être ajoutés si on écrit une spécification (cf. RFC 8126). Notez que chaque espace a son UUID (6ba7b810-9dad-11d1-80b4-00c04fd430c8 pour l'espace DNS).

Les UUID de version 6 et 7, nouveautés de ce RFC 9562, ne sont pas mis en œuvre par uuidgen, ni d'ailleurs par beaucoup d'autres programmes.

Les sections 6.1 et 6.2, elles, décrivent le processus de génération d'un UUID à base temporelle. Idéalement, il faut utiliser une graine enregistrée sur le disque (pour éviter de générer des UUID identiques) ainsi que l'instant de la génération. Mais lire sur le disque prend du temps (alors qu'on peut vouloir générer des UUID rapidement, par exemple pour identifier des transactions) et l'horloge de la machine n'a pas toujours une résolution suffisante pour éviter de lire deux fois de suite le même instant. Ces sections contiennent donc également des avis sur la génération fiable d'UUID, par exemple en gardant en mémoire le nombre d'UUID générés, pour les ajouter à l'heure.

La section 8, consacrée à la sécurité, rappelle qu'un UUID ne doit pas être utilisé comme capacité (car il est trop facile à deviner) et qu'il ne faut pas demander à un humain de comparer deux UUID (ils se ressemblent trop pour un œil humain).

Il est évidemment recommandé d'utiliser les UUID de version 5 plutôt que de version 3 (RFC 6151) mais SHA-1 a aussi ses problèmes (RFC 6194) et, de toute façon, pour la plupart des utilisations des UUID, les faiblesses cryptographiques de MD5 et de SHA-1 ne sont pas gênantes.

La section 2.1 du RFC détaille les motivations pour la mise à jour du RFC 4122 et quels sont les changements effectués. Certaines utilisations des UUID ont remis en cause des suppositions originales. Ainsi, les UUID sont souvent utilisés dans un contexte réparti, où leur capacité à être uniques sans registre central est très utile. Mais quelques points manquaient au RFC 4122 :

— UUID version 4 (aléatoire) avait une mauvaise localité : deux UUID créés l'un après l'autre en un temps très court n'ont aucun rapport, ce qui est gênant pour certains usages (par exemple comme étiquette dans un "*B-tree*").
— UUID version 1 (fondé entre autre sur le temps écoulé depuis l'"*epoch*") utilise un incrément peu pratique (cent nanosecondes).
— Certaines méthodes fabrication des UUID posaient des gros problèmes de vie privée, par exemple l'utilisation des adresses MAC dans UUID version 1, désormais déconseillée.
— Le RFC 4122 descendait trop dans les détails de mise en œuvre.
— Le RFC 4122 ne séparait pas les exigences pour la génération d'UUID et celles pour leur stockage.

Seize mises en œuvre des UUID ont été étudiées pour préparer le nouveau RFC (vous avez la liste dans la section 2.1), menant aux constations suivantes :

— Beaucoup d'errata <https://www.rfc-editor.org/errata/rfc4122> dans le RFC 4122.
— Spécification du format qui mélangeait trop les diverses versions d'UUID.
— Absence de vecteurs de test (ils figurent désormais dans l'annexe A).

En Python, il existe un module UUID <https://docs.python.org/3/library/uuid.html> qui offre des fonctions de génération d'UUID de différentes versions (mais pas les plus récentes) :

```
import uuid

myuuid = uuid.uuid1() # Version 1, Time-based UUID
heruuid = uuid.uuid3(uuid.NAMESPACE_DNS, "foo.bar.example") # Version
            # 3, Name-based ("hash-based") UUID, a name hashed by MD5
otheruuid = uuid.uuid4() # Version 4, Random-based UUID
yetanotheruuid = uuid.uuid5(uuid.NAMESPACE_DNS,
                            "www.example.org")
                        # Version 5, a name hashed by SHA1
if (myuuid == otheruuid or \
    myuuid == heruuid or \
    myuuid == yetanotheruuid or \
    otheruuid == yetanotheruuid):
    raise Exception("They are equal, PANIC!")

print(myuuid)
print(heruuid) # Will always be the same
print(otheruuid)
print(yetanotheruuid) # Will always be the same
```

Et comme le dit la documentation, *"Note that uuid1() may compromise privacy since it creates a UUID containing the computer[Caractère Unicode non montré [2]]s network address."* (méthode de génération des UUID version 1 qui est désormais déconseillée).

Le SGBD PostgreSQL inclut un type UUID <https://www.postgresql.org/docs/current/datatype-uuid.html>. Cela évite de stocker les UUID sous leur forme texte, ce qui est techniquement absurde et consomme 288 bits au lieu de 128 (section 6.13 du RFC).

```
essais=> CREATE TABLE Transactions (id uuid, value INT);
CREATE TABLE
essais=> INSERT INTO Transactions VALUES
           ('74738ff5-5367-5958-9aee-98fffdcd1876', 42);
INSERT 0 1
essais=> INSERT INTO Transactions VALUES
            ('88e6441b-5f5c-436b-8066-80dca8222abf', 6);
INSERT 0 1
essais=> INSERT INTO Transactions VALUES ('Pas correct', 3);
ERROR:  invalid input syntax for type uuid: "Pas correct"
LINE 1: INSERT INTO Transactions VALUES ('Pas correct', 3);
                                         ^
-- PostgreSQL peut seulement générer la version 4, les aléatoires
essais=> INSERT INTO Transactions VALUES (gen_random_uuid () , 0);
INSERT 0 1
essais=> SELECT * FROM Transactions;
                  id                  | value
--------------------------------------+-------
 74738ff5-5367-5958-9aee-98fffdcd1876 |    42
 88e6441b-5f5c-436b-8066-80dca8222abf |     6
 41648aef-b123-496e-8a4c-52e573d17b6a |     0
(3 rows)
```

Attention, le RFC (section 6.13) déconseille l'utilisation des UUID fondés sur un nom pour servir de clé primaire dans la base de données, sauf si on est absolument certain (mais c'est rare) que les noms ne changeront pas.

Il existe plusieurs exemples d'utilisation des UUID. Par exemple, Linux s'en sert pour identifier les disques attachés à la machine, de préférence à l'ancien système où l'ajout d'un nouveau disque pouvait changer l'ordre des numéros sur le bus et empêcher le système de trouver un disque. Un `/etc/fstab` typique sur Linux contient donc des :

2. Car trop difficile à faire afficher par LaTeX

```
UUID=da8285a0-3a70-413d-baed-a1f48d7bf7b2       /home   ext3 defaults ...
```

plutôt que les anciens :

```
/dev/sda3       /home           ext3    defaults
```

car `sda3` n'est pas un identificateur stable. L'UUID, lui, est dans le système de fichiers et ne changera pas avec les changements sur le bus. On peut l'afficher avec `dumpe2fs` :

```
# dumpe2fs -h /dev/sda3
...
Filesystem UUID:          da8285a0-3a70-413d-baed-a1f48d7bf7b2
...
```

Un exemple d'utilisation de la nouvelle version 7 est décrit dans l'excellent article « "*Goodbye integers. Hello UUIDv7!*" <https://buildkite.com/blog/goodbye-integers-hello-uuids> ». Une mise en œuvre de cette version 7 apparait dans « "*UUIDv7 in 20 languages*" <https://antonz.org/uuidv7/> ».

La section 6 décrit les bonnes pratiques de génération d'un UUID. Ainsi :

— Pour tous les UUID fondés sur le temps, si vous avez besoin d'unicité des UUID, attention aux cas où l'horloge peut changer, en raison d'une modification manuelle, d'une seconde intercalaire ou d'un ajustement fait par NTP.
— Si vous générez des UUID de manière répartie, sur plusieurs machines, ce qui est un des points forts des UUID, mais que vous voulez quand même de l'unicité, vous pouvez préfixer vos UUID avec un identificateur de la machine, ou bien compter sur un registre central (mais, en général, c'est justement ce qu'on veut éviter avec des UUID).
— UUID rend les collisions (génération de deux UUID identiques) peu probables mais pas impossibles. Lorsque vous concevez une application, demandez-vous si une collision est juste un petit inconvénient ou bien si elle est inacceptable (le RFC cite l'exemple fictif d'un système de contrôle aérien où chaque avion serait identifié par un UUID ; l'attribution du même UUID à deux avions différents serait clairement catastrophique). Dans ce cas, vous devez trouver un moyen de ne pas avoir de collision.
— Si vous voulez des UUID qui ne soient pas prévisibles par un observateur extérieur, utilisez la version 4 et assurez vous que votre générateur de nombres aléatoires suit bien les prescriptions des RFC 4086 et RFC 8937, ainsi que « "*Random Number Generator Recommendations for Applications*" <https://peteroupc.github.io/random.html> ».
— Les versions 6 et 7 (nouveautés de ce RFC 9562) sont prévues pour que les UUID soient triables chronologiquement, sans avoir besoin d'analyse, uniquement en triant la version binaire. Un des avantages est que des UUID générés à des moments proches ont des valeurs proches, ce qui peut être utile dans certaines applications.