On the ways of creating a GUID that looks pretty

devblogs.microsoft.com/oldnewthing/20150410-00

April 10, 2015

Raymond Chen

A customer had what at first appeared to be a question born of curiousity.

> Is it possible that a GUID can be generated with all ASCII characters for bytes? In other words, is it possible that a GUID can be generated, and then if you interpret each of the 16 bytes as an ASCII character, the result is a printable string? Let's say for the sake of argument that the printable ASCII characters are U+0020 through U+007E.

Now, one might start studying the various GUID specifications to see whether such as GUID is legal. For example, types 1 and 2 are based on a timestamp and MAC address. An all-ASCII MAC address is legal. The locally-administered bit has value 0x02, and one you set that bit, all the other bits can be freely assigned by the network administrator. But then you might notice the Type Variant field, and the requirement that all new GUIDs must set the top bit, so that takes you out of the printable ASCII region, so bzzzzt, no all-ASCII GUID for you.

But we've fallen into the trap of answering the question instead of solving the problem.

What is the problem that you're trying to solve, where you are wondering about all-ASCII GUIDs?

> We want to create some sentinel values in our database, and we figured we could use some all-ASCII GUIDs for convenience.

If you want a sentinel value that is guaranteed to be unique, why not create a GUID?

```
C:\> uuidgen
GUID_SpecialSentinel = {that GUID}
```

Now you are guaranteed that the value is unique and will never collide with any other valid GUID.

> We could do that, but we figured it'd be handy if those sentinel values spelled out something so they'd be easier to spot in a dump file. If we know that all-ASCII GUIDs are not valid, then we can use all-ASCII GUIDs for our sentinel values.

Now, while `uuidgen` does produce valid GUIDs, it's also the case that those valid GUIDs aren't particularly easy to remember, nor do they exactly trip off the tongue. After all, the space of values that are easy to pronounce and remember is much, much smaller than 2^{128}. It's probably more on the order of 2^{20}, which is not enough bits to ensure global uniqueness. Heck, it's not even enough bits to describe all the pixels on your screen!

So woot! Since all-ASCII GUIDs are not generatable under the current specification for GUIDs, I can go ahead and name my GUID `{6d796152-6e6f-4364-6865-6e526f636b73}` which shows up in a memory dump as

```
52 61 79 6d 6f 6e 64 43-68 65 6e 52 6f 63 6b 73  RaymondChenRocks
```

I am so awesome!

But even if you convince yourself that no current GUID generation algorithm could create a GUID that collides with your special easy-to-remember and quick-to-pronounce sentinel GUIDs, there is no guarantee that you will make a particularly unique choice of sentinel value.

This is also known as What if two people did this?

There are many people named Raymond Chen in the world. Heck, there are many people named Raymond Chen *at Microsoft*. (We get each other's mail sometimes.) What if somebody else named Raymond Chen gets this same clever idea and creates their own sentinel value called `RaymondChenRocks`? Everything works great until my database starts interoperating with the other Raymond Chen's database, and now we have a GUID collision.

Now, the most common way to create a duplicate GUID is to duplicate it. But here, we created a duplicate GUID because the thing we created *was not generated via a duplicate-avoidance algorithm*. If the algorithm wasn't designed to avoid duplicates, then it's not too surprising that there may be duplicates. I just pulled this GUID out of my butt. (Mind you, my butt rocks.)

Okay, so let's go back to the original problem so we can solve it.

The most straightforward solution is simply to create a standard GUID each time you need a new sentinel value. "Oh, I need a GUID to represent an item which has been discontinued. Let me run uuidgen and hey look, there's a new GUID. I will call it `GUID_Discontinued`." This solves the uniqueness problem, and it is very simple to explain and prove correct. This is what people end up doing the vast majority of the time, and it's what I recommend.

Okay, you want to have the property that these special GUIDs can be easily spotted in crash dumps. One way to do this is to extract the MAC address from a network card, then destroy the card. You can now use the 60 bits of the timestamp fields to encode your ASCII message.

A related problem is that you want to generate a GUID based on some other identifying information, with the properties that

- Two items with the same identifying information should have the same GUID.
- Two items with different identifying information should have different GUIDs.
- None of these GUIDs should collide with GUIDs generated by any other means.

For that, you can use a name-based GUID generation algorithm.

Raymond Chen

Follow