GUIDs are globally unique, but substrings of GUIDs aren’t

devblogs.microsoft.com/oldnewthing/20080627-00

June 27, 2008

Raymond Chen

A customer needed to generate an 8-byte unique value, and their initial idea was to generate a GUID and throw away the second half, keeping the first eight bytes. They wanted to know if this was a good idea.

No, it’s not a good idea.

The GUID generation algorithm relies on the fact that it has all 16 bytes to use to establish uniqueness, and if you throw away half of it, you lose the uniqueness. There are multiple GUID generation algorithms, but I’ll pick one of them for concreteness, specifically the version described in this Internet draft.

The first 60 bits of the GUID encode a timestamp, the precise format of which is not important.

The next four bits are always 0001, which identify that this GUID was generated by “algorithm 1”. The version field is necessary to ensure that two GUID generation algorithms do not accidentally generate the same GUID. The algorithms are designed so that a particular algorithm doesn’t generate the same GUID twice, but without a version field, there would be no way to ensure that some other algorithm wouldn’t generate the same GUID by some systematic collision.

The next 14 bits are “emergency uniquifier bits”; we’ll look at them later, because they are the ones that fine tune the overall algorithm.

The next two bits are reserved and fixed at 01.

The last 48 bits are the unique address of the computer’s network card. If the computer does not have a network card, set the top bit and use a random number generator for the other 47. No valid network card will have the top bit set in its address, so there is no possibility that a GUID generated from a computer without a network card will accidentally collide with a GUID generated from a computer *with* a network card.

Once you take it apart, the bits of the GUID break down like this:

- 60 bits of timestamp,
- 48 bits of computer identifier,
- 14 bits of uniquifier, and
- six bits are fixed,

for a total of 128 bits.

The goal of this algorithm is to use the combination of time and location ("space-time coordinates" for the relativity geeks out there) as the uniqueness key. However, timekeeping is not perfect, so there's a possibility that, for example, two GUIDs are generated in rapid succession from the same machine, so close to each other in time that the timestamp would be the same. That's where the uniquifier comes in. When time appears to have stood still (if two requests for a GUID are made in rapid succession) or gone backward (if the system clock is set to a new time earlier than what it was), the uniquifier is incremented so that GUIDs generated from the "second time it was five o'clock" don't collide with those generated "the first time it was five o'clock".

Once you see how it all works, it's clear that you can't just throw away part of the GUID since all the parts (well, except for the fixed parts) work together to establish the uniqueness. If you take any of the three parts away, the algorithm falls apart. In particular, keeping just the first eight bytes (64 bits) gives you the timestamp and four constant bits; in other words, all you have is a timestamp, not a GUID.

Since it's just a timestamp, you can have collisions. If two computers generate one of these "truncated GUIDs" at the same time, they will generate the same result. Or if the system clock goes backward in time due to a clock reset, you'll start regenerating GUIDs that you had generated the first time it was that time.

Upon further investigation, the customer really didn't need global uniqueness. The value merely had to be unique among a cluster of a half dozen computers. Once you understand *why* the GUID generation algorithm works, you can reimplement it on a smaller scale:

- Four bits to encode the computer number,
- 56 bits for the timestamp, and
- four bits as a uniquifier.

We can reduce the number of bits to make the computer unique since the number of computers in the cluster is bounded, and we can reduce the number of bits in the timestamp by assuming that the program won't be in service 200 years from now, or that if it is, the items that were using these unique values are no longer relevant. At 100 nanoseconds per tick, 2^56 ticks will take 228 years to elapse. (Extending the range beyond 228 years is left as an exercise, but it's wasted effort, because you're going to hit the 16-computer limit first!)

You can get away with a four-bit uniquifier by assuming that the clock won't drift more than an hour out of skew (say) and that the clock won't reset more than sixteen times per hour. Since you're running under a controlled environment, you can make this one of the rules for running your computing cluster.

Raymond Chen

Follow