Deterministic Guid Generation

History

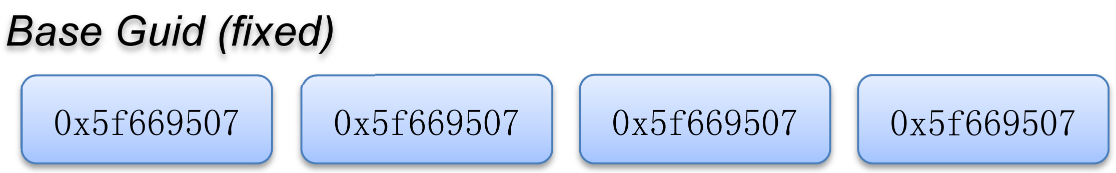
v0.1.2	2013-07-26	Multiple random seed explanation added
v0.1.1	2013-07-11	history added, style and format improved
v0.1	2013-07-05	created by Gu Lu

Goal

1) It should have the same level of collision probability as current pure COM guid generation.
   - Existing code logic heavily depends on this requirements so we shouldn't break it at all cost
2) It should be deterministic and consistent.
   - Easier to identify and isolate changes to content file
   - Make the cooked content file reusable

Method (three-step process)

1) We use a pre-generated guid as a "base guid"

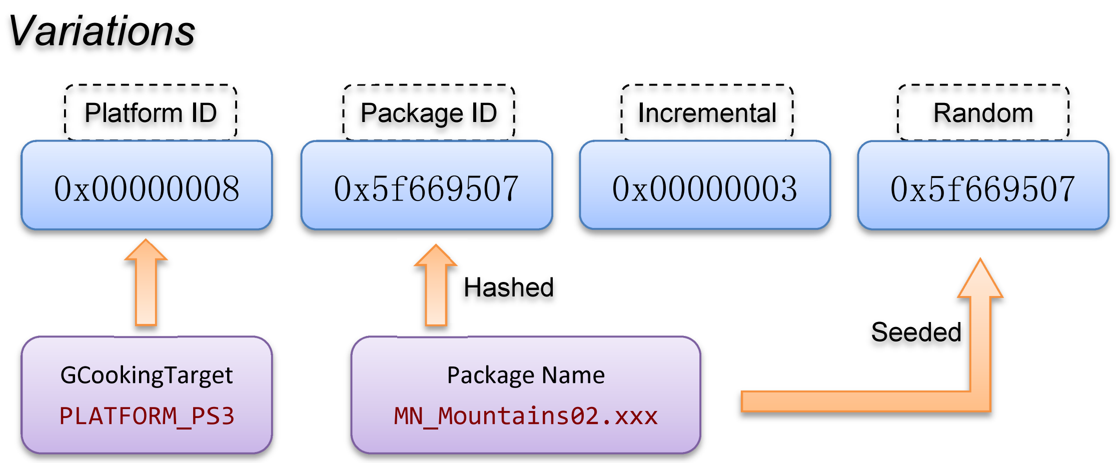


a) This ensures the uniqueness of our later variations. Theoretically, if the "base guid" is unique to all existing guids, all derived variations should have more or less the same level of collision probabilities as the "base guid".

b) The "base guid" acts as a "finger print" to all derivations. Once we have the same key, we can ensure that all generated derivations are compatible to each other.

c) When we want to invalidate all existing contents (it might sound useless), you can just switch the "base guid".

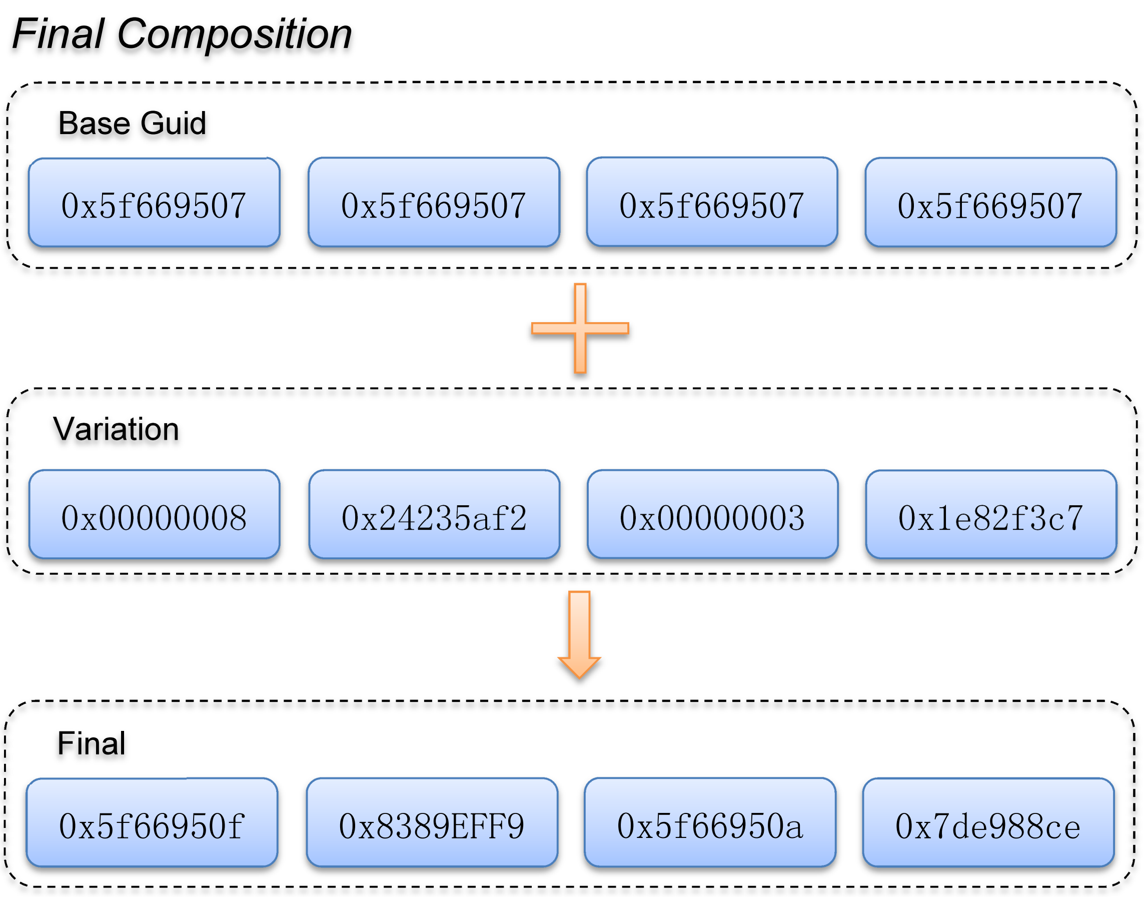
2) We use a package-based generation of multiple random sequences as variations.



a) We are using the hashed name of current package as the random seed to ensure that any time we cook the same package we get the same result.
b) The random sequences are thus independent to each other among packages, so any future change to given existing package won't spread into other packages.
c) An incremental number is encoded so they won't collide even if two same numbers are generated by the random generator.
d) The target platform id is also encoded to prevent colliding among different platforms.

3) Combine the "base guid" and the "variation"

By mixing a real "base guid" and multiple seeded random sequences, we get a guid which preserves the same level of uniqueness and is reproducible and consistent.



Notes, requirements and potential issues

Multiple random seed

Implemented using this C++11 constructs

```
    std::tr1::mt19937 RandGenerator;
    std::tr1::uniform_int<DWORD> RandDistribution;
```

Local shader cleanup (Now has been automated)

Local shader cache should be cleaned and generated from the ground up each time to prevent out-of-order guid generation.

```
    // this operation is performed after appInit(),
    // since GetLocalShaderCacheFilename() requires GConfig being initialized.
#if WITH_DUST && !FINAL_RELEASE
    if (appStristr(appCmdLine(), TEXT("consistent_mode")) != NULL)
    {
        GFileManager->Delete(*GetLocalShaderCacheFilename(SP_PCD3D_SM3));
        GFileManager->Delete(*GetLocalShaderCacheFilename(SP_PS3));
    }
#endif
```

Deprecated alternative

We can make appCreateGuid() class based, which means each UObject-derived class has its own random generator (using hashed class name as seed). But this method is relatively heavy, since it turns appCreateGuid() into a template function which instantiates by specific UObject-derived class. This also needs to change every caller, which is a bit cumbersome and irrational.