Support for "global" engines in `SeedService`

Gianluca Petrillo

Fermi National Accelerator Laboratory

LArSoft Coordinators' Meeting, April 12th , 2016

Random number generators recap

`art::RandomNumberGenerator` is a *art* service that:

- creates and manages random number engines
- only supports engines derived from `CLHEP::HepRandomEngine`
- in fact, it only supports the engines it supports (not user-extensible)
- binds each engine to a specific module instance
- does not conceive engines not related to any module instance

The framework can help help with restoring the state of these engines.

While LArSoft strongly recommends your engines to be managed with `RandomNumberGenerator`, "unmanaged" engines can coexist — they lose some functionality (**and make debugging a major pain**)

What is a *random engine*?

The same as "random number generator": any piece of code generating a sequence of pseudo-random numbers uniformly distributed in $[0, 1[$.

`Seed`what??

`artext::SeedService` is a service in artextensions that:

- provides central control of random engine seeds
- supports a few different "policies" to automatically assign seeds to engines
- allows the user to override single seeds
- accepts "any" type of engine (you pay the interface)
- binds each seed and engine to a specific module instance

While LArSoft strongly recommends your seeds to be managed with `SeedService`, engines with "unmanaged" seeds can coexist — they lose some more functionality (**and make debugging a even major pain**)

`SeedService` interface

How to use it:

1. ask `SeedService` to get you a seed for an engine
2. create an engine with `createEngine()` (that calls `RandomNumberGenerator`): must be in the module constructor
3. tell `SeedService` about this engine

or:

1. ask `SeedService` to create an engine (that calls module's `createEngine()`, that calls `RandomNumberGenerator`) and set its seed for you: again, must be in the module constructor

or:

1. create your own engine
2. ask `SeedService` to register its presence (sometimes not trivial) and sets its seed: and again, must be in the module constructor

Later, you can ask `SeedService` which was the actual seed.

New feature: “global engine”

Feature request #11125 asks `SeedService` to support engines that are not bound to a module. *(I’ll call them “global engines” for lack of imagination)*
The current limitations are in place:

- to guarantee the functionality of per-event seed policies, all engines must be “registered” to the service
- as a safety measure, registration is only allowed during construction: when modules are done being constructed, registration is closed
- within module constructors, only module-bound engines can be created
- outside module constructors, only global engines can be created
- no life management is provided for global engines either: you create it, you own it, you destroy it

In fact, the use of the current interface to ask the module to create an engine is discouraged, the recommended way being for the module to directly create the engine via framework.

Example: engine owned by a service

For example, in a service constructor:

```
// create a new engine, that we own (engine be a class data member)
engine = std::make_unique<TRandom3>();
art::ServiceHandle<artext::SeedService>()->registerEngine
  (artext::SeedService::TRandomSeeder(engine.get()), "myService");

// MyService callback registrations in the ActivityRegistry
// should follow the first call to SeedService!
```

The engine instance name (`"myService"`) must be unique in the whole job *(and refrain from calling it `"gaushit"`...)*.
When registering an engine that is not a `CLHEP::HepRandomEngine`, as in this example, a “seeder” must be provided (the documentation explains how).

Example: per-engine seed configuration

Many policies do not require specific configuration.
FHiCL configuration for policies that require a seed or offset:

```
services.SeedService: {
   policy             : "preDefinedSeed"
   endOfJobSummary    :  true

   myModule:  7
   myService: 2
}
```

Status and summary

- artextensions code is ready in a feature branch
- no LArSoft adaption has been tried yet
  (actually, no change is expected to be needed)
- complete documentation is in Doxygen format in the source,
  `SeedService.hh`
  (but currently LArSoft's Doxygen is not picking it up)
- wiki documentation will be updated once changes are accepted

My two cents

Think well before having a service own an engine: random engines are a multi-thread pain, and global ones more so (also known as: services are better designed stateless).
If you can, prefer modules to pass your service the engine to be used.