Avoid discarding createEngine's returned reference, or

Down with art's getEngine!

Kyle J. Knoepfel

LArSoft Coordination Meeting

29 January 2019

Random number engines in *art*/LArSoft

- *art* provides random-number management via the `RandomNumberGenerator` service.
  - Engines are attributed to modules, using the `createEngine(…)` base-class member function.
    - To access the engine, just cache the returned reference from the `createEngine` call

```
CLHEP::HepRandomEngine& engine = createEngine(…);
```

- *Directly using the `RandomNumberGenerator` service has rarely been required.*

Random number engines in *art*/LArSoft

- *art* provides random-number management via the `RandomNumberGenerator` service.
  - Engines are attributed to modules, using the `createEngine(…)` base-class member function.
    - To access the engine, just cache the returned reference from the `createEngine` call

```
CLHEP::HepRandomEngine& engine = createEngine(…);
```

- *Directly using the `RandomNumberGenerator` service has rarely been required.*
- For nutools users, the `NuRandomService` has its own `createEngine` function:
  - The `NuRandomService::createEngine` function wraps the *art*-provided function.
  - Until recently, the only way you could gain access to the engine was to call *art*'s `RandomNumberGenerator::getEngine` function.

Random number engines in *art*/LArSoft

- *art* provides random-number management via the `RandomNumberGenerator` service.
  - Engines are attributed to modules, using the `createEngine(…)` base-class member function.
    - To access the engine, just cache the returned reference from the `createEngine` call

```
CLHEP::HepRandomEngine& engine = createEngine(…);
```

- *Directly using the `RandomNumberGenerator` service has rarely been required.*
- For nutools users, the `NuRandomService` has its own `createEngine` function:
  - The `NuRandomService::createEngine` function wraps the *art*-provided function.
  - Until recently, the only way you could gain access to the engine was to call *art*'s `RandomNumberGenerator::getEngine` function.

`RNG::getEngine` will be deprecated in *art* 3.02, removed in *art* 3.03

Problems with `getEngine` (*art* 2)

- It's confusing.
- Which engine does the following code use?

```
double random_number( )
{
  // Common to see in art 2
  ServiceHandle<RNG> rng;
  auto& engine = rng->getEngine( );
  CLHEP::RandFlat rand{engine};
  return rand.fire( );
}
```

Problems with `getEngine` (*art* 3)

- It's confusing
- Which engine does the following code use?

```
double random_number( )
{
  // Common to see in art 3
  ServiceHandle<RNG> rng;
  auto& engine = rng->getEngine(module_label, // Where do you get
                                schedule_id); // these values?
  CLHEP::RandFlat rand{engine};
  return rand.fire( );
}
```

Problems with `getEngine`

Common

```
class MyProducer {
public:

  MyProducer(ParameterSet const& pset)
  {
    ServiceHandle<NuRandomService>{}
      ->createEngine(...);
  }

  void produce(art::Event& e) override
  {
    auto& engine =
      ServiceHandle<RandomNumberGenerator>{}
        ->getEngine(...);
    CLHEP::RandFlat flatDist{engine};
    flatDist.fire(...);
  }
};
```

Problems with `getEngine`

Common

```
class MyProducer {
public:

  MyProducer(ParameterSet const& pset)
  {
    ServiceHandle<NuRandomService>{}
      ->createEngine(...);
  }

  void produce(art::Event& e) override
  {
    auto& engine =
      ServiceHandle<RandomNumberGenerator>{}
        ->getEngine(...);
    CLHEP::RandFlat flatDist{engine};
    flatDist.fire(...);
  }
};
```

Expensive operations:

- *`ServiceHandle` created for each event*
- *`getEngine` called on each event*
- *`RandFlat` distribution created for each event*

What's the better option?

Common

```
class MyProducer {
public:

  MyProducer(ParameterSet const& pset)
  {
    ServiceHandle<NuRandomService>{}
      ->createEngine(...);
  }

  void produce(art::Event& e) override
  {
    auto& engine =
      ServiceHandle<RandomNumberGenerator>{}
        ->getEngine(...);
    CLHEP::RandFlat flatDist{engine};
    flatDist.fire(...);
  }
};
```

Better

```
class MyProducer {
  CLHEP::RandFlat flatDist_;

public:

  MyProducer(ParameterSet const& pset)
    : flatDist_{ServiceHandle<NuRandomService>{}
                  ->createEngine(...)}
  {}

  void produce(art::Event& e) override
  {
    flatDist_.fire(...);
  }
};
```

What's the better option?

Common

```
class MyProducer {
public:

  MyProducer(ParameterSet const& pset)
  {
    ServiceHandle<NuRandomService>{}
      ->createE
  }

  void produce(art::Event& e) override
  {
    auto& engine =
      ServiceHandle<RandomNumberGenerator>{}
        ->getEngine(...);
    CLHEP::RandFlat flatDist{engine};
    flatDist.fire(...);
  }
};
```

createEngine returns art-owned reference to engine; no need to directly interact with it

Better

```
class MyProducer {
  CLHEP::RandFlat flatDist_;

public:

  MyProducer(ParameterSet const& pset)
    : flatDist_{ServiceHandle<NuRandomService>{}
                  ->createEngine(...)}
  {}

  void produce(art::Event& e) override
  {
    flatDist_.fire(...);
  }
};
```

What's the better option?

Common

```
class MyProducer {
public:

  MyProducer(ParameterSet const& pset)
  {
    ServiceHandle<NuRandomService>{}
      ->createEngine(...);
  }

  void produce(art::Event& e) override
  {
    auto& engine =
      ServiceHandle<RandomNumberGenerator>{}
        ->getEngine(...);
    CLHEP::RandFlat flatDist{engine};
    flatDist.fire(...);
  }
};
```

Better

```
class MyProducer {
  CLHEP::RandFlat flatDist_;

public:

  MyProducer(ParameterSet const& pset)
    : flatDist_{ServiceHandle<NuRandomService>{}
                  ->createEngine(...)}
  {}

  void produce(art::Event& e) override
  {
    flatDist_.fire(...);
  }
};
```

You can do this now.

To encourage better usage…

- I would like to make it a compile-time warning to discard the reference returned when calling `NuRandomService::createEngine`.

```
// The following statement is supported:
engine_t& engine = ServiceHandle<NuRandomService>{}->createEngine(...);

// The following statement generates a compile-time warning:
ServiceHandle<NuRandomService>{}->createEngine(...);
```

To encourage better usage…

- I would like to make it a compile-time warning to discard the reference returned when calling `NuRandomService::createEngine`.

```
// The following statement is supported:
engine_t& engine = ServiceHandle<NuRandomService>{}->createEngine(...);

// The following statement generates a compile-time warning:
ServiceHandle<NuRandomService>{}->createEngine(...);
```

```
/home/knoepfel/scratch/larsoft-devel/srcs/larsim/larsim/EventGenerator/LightSource_module.cc:198:83:
warning: ignoring return value of ‘std::reference_wrapper<CLHEP::HepRandomEngine>
rndm::NuRandomService::createEngine(…) […]’, declared with attribute nodiscard [-Werror=unused-result]
     art::ServiceHandle<rndm::NuRandomService>{}->createEngine(*this, pset, "Seed2");
```

To encourage better usage…

- I would like to make it a compile-time warning to discard the reference returned when calling `NuRandomService::createEngine`.

```
// The following statement is supported:
engine_t& engine = ServiceHandle<NuRandomService>{}->createEngine(...);

// The following statement generates a compile-time warning:
ServiceHandle<NuRandomService>{}->createEngine(...);
```

```
/home/knoepfel/scratch/larsoft-devel/srcs/larsim/larsim/EventGenerator/LightSource_module.cc:198:83:
warning: ignoring return value of ‘std::reference_wrapper<CLHEP::HepRandomEngine>
rndm::NuRandomService::createEngine(…) […]’, declared with attribute nodiscard [-Werror=unused-result]
     art::ServiceHandle<rndm::NuRandomService>{}->createEngine(*this, pset, "Seed2");
```

- In rare circumstances, the warning can be silenced by casting to void:

```
(void)ServiceHandle<NuRandomService>{}->createEngine(...);
```

To encourage better usage…

- I would like to make it a compile-time warning to discard the reference returned when calling `NuRandomService::createEngine`.

```
// The following statement is supported:
engine_t& engine = ServiceHandle<NuRandomService>{}->createEngine(...);

// The following statement generates a compile-time warning:
ServiceHandle<NuRandomService>{}->createEngine(...);
```

```
/home/knoepfe[illegible].cc:198:83:
warning: igno[illegible]
rndm::NuRando[illegible]nused-result]
     art::Ser[illegible]
```

Making this change now will position LArSoft for *art* 3.02, when getEngine is deprecated by *art*.

- In rare circumstances, the warning can be silenced by casting to void:

```
(void)ServiceHandle<NuRandomService>{}->createEngine(...);
```

The proposal

- Make it a compile-time warning to discard the reference returned when calling `NuRandomService::createEngine`.

- Under-the-covers technical change:

```
- engine_t& createEngine(...)
+ [[nodiscard]] std::reference_wrapper<engine_t> createEngine(...)
```

- Feature branches available:
  - `nutools:feature/knoepfel_nodiscard`
  - `larsim:feature/knoepfel_nodiscard`
  - `larreco:feature/knoepfel_nodiscard`
  - `larana:feature/knoepfel_nodiscard`

- Will provide feature branches for experiments if the above change is accepted.