# *art* news

Kyle J. Knoepfel

17 May 2018

# `RandomNumberGenerator` issues for *art* 3

- The RandomNumberGenerator service provides random-number engine management for *art* users.
  - *art* owns the engines; users own the distributions.



Fermilab

# `RandomNumberGenerator` issues for *art* 3

- The RandomNumberGenerator service provides random-number engine management for *art* users.
  - *art* owns the engines; users own the distributions.

```
class RNGProducer : public art::EDProducer {
public:
  explicit RNGProducer(Parameters const& p)
    : dist_{createEngine(...)}
  {}
private:
  void analyze(art::Event const&) override {
    dist_.fireInt(...);
  }
  CLHEP::RandFlat dist_;
}
```



Fermilab

# `RandomNumberGenerator` issues for *art* 3

- The RandomNumberGenerator service provides random-number engine management for *art* users.
  - *art* owns the engines; users own the distributions.

```
class RNGProducer : public art::EDProducer {
public:
  explicit RNGProducer(Parameters const& p)
    : dist_{createEngine(...)}
  {}
private:
  void analyze(art::Event const&) override {
    dist_.fireInt(...);
  }
  CLHEP::RandFlat dist_;
}
```

**Reference to *art*-owned engine**

**Module-owned**



Fermilab

# `RandomNumberGenerator` issues for *art* 3

- The RandomNumberGenerator service provides random-number engine management for *art* users.
  - *art* owns the engines; users own the distributions.

```
class RNGProducer : public art::EDProducer {
public:
  explicit RNGProducer(Parameters const& p)
    : dist_{createEngine(...)}
  {}
private:
  void analyze(art::Event const&) override {
    dist_.fireInt(...);
  }
  CLHEP::RandFlat dist_;
}
```

**Reference to *art*-owned engine**

**Module-owned**

**No need to create a service handle.**

**This is how it should be.**



Fermilab

# Enter `ServiceHandle<RandomNumberGenerator>`

- Through *art* 2, RandomNumberGenerator has had a concept of the "current" module being processed:

```
ServiceHandle<RNG>{}->getEngine();
ServiceHandle<RNG>{}->getEngine("the_other_one");
```

- In *art* 3, there is no longer any "current" module. The equivalent interface is:

```
ServiceHandle<RNG> rng{};
rng->getEngine(scheduleID, moduleLabel);
rng->getEngine(scheduleID, moduleLabel, "the_other_one");
```

- If all engines are retrieved using the `createEngine` interface, then `getEngine` can be removed, and the correct engine can be given by reference to the functions that need it.
  - Direct access to the `RandomNumberGenerator` service is no longer needed.



Fermilab

# A necessary change regardless

- Removing the concept of a “current” module implies a change to how modules are constructed/`createEngine`(…) is used:

```
// art 2
RNGProducer(Parameters const& p)
  : dist_{createEngine(p().seed())}
{}

// art 3
RNGProducer(Parameters const& p)
  : art::EDProducer{p} // must specify base class c’tor
  , dist_{createEngine(p().seed())}
{}
```



Fermilab

# getMany(ByType) behavior

- Consider the following configuration:

```
physics: {
  producers: {
    make1: { module_type: IntMaker  value: 1 }
    make7: { module_type: IntMaker  value: 7 }
    readIntsThenMakeSomeMore: { module_type: MoreInts }
  }
  p1: [make1, make7]
  p2: [readIntsThenMakeSomeMore] // relies on getMany
}
```

- According to *art*'s path-execution guarantees, the products made in p1 may not be available whenever p2 is executed.



Fermilab

# getMany(ByType) behavior

- Consider the following configuration:

```
physics: {
  producers: {
    make1: { module_type: IntMaker  value: 1 }
    make7: { module_type: IntMaker  value: 7 }
    readIntsThenMakeSomeMore: { module_type: MoreInts }
  }
  p1: [make1, make7]
  p2: [readIntsThenMakeSomeMore] // relies on getMany
}
```

- According to *art*'s path-execution guarantees, the products made in p1 may not be available whenever p2 is executed.
- In MT execution, there are no trigger-path ordering guarantees at all.



Fermilab

# `getMany(ByType)` behavior

- Consider the following configuration:

**Options for *art* 3:**

1. Retain unspecified (and nondeterministic in MT) behavior.
2. Remove `getMany*` interface.
3. For producers and filters, disable use of `getMany*` (at run-time).
4. For all modules, getMany* can only retrieve products from the input source.
5. For producers and filters, `getMany*` can only retrieve products from the input source.

- According to *art*'s path-execution guarantees, the products made in `p1` may not be available whenever `p2` is executed.
- In MT execution, there are no trigger-path ordering guarantees at all.



Fermilab