



## Multi-threaded *art* forum – *art* 3.0

Kyle J. Knoepfel

1 March 2018

# For today

- **Review**
- **art 3.0 (proposed)**
  - General support
  - Secondary input files
  - Product mixing
  - TFileService constraints
  - Data-product dependency checking
  - Replicated-module reduction facility
- **Next steps**

# For today

- **Review**
- **art 3.0 (proposed)**
  - General support
  - Secondary input files
  - Product mixing
  - TFileService constraints
  - Data-product dependency checking
  - Replicated-module reduction facility
- **Next steps**

***We have questions and we need your answers!***

# art MT forum – Session 1

[https://cdcvs.fnal.gov/redmine/projects/art/wiki/Art\\_multi-threading\\_forum\\_-\\_introduction](https://cdcvs.fnal.gov/redmine/projects/art/wiki/Art_multi-threading_forum_-_introduction)

- We discussed motivations for a multi-threaded (MT) framework
- Largely based off of CMSSW’s design
  - We use Intel’s Threading Building Blocks (TBB)
  - Steps to be performed are factorized into *tasks*
  - You can think of a call to your module’s “produce” function as performing a task
- Users specify the number of concurrent schedules (i.e. event loops) and (optionally) the maximum number of threads that the process can use.
- Each loop processes one event at a time.
- Different modules will also be able to be run in parallel on the *same* event.

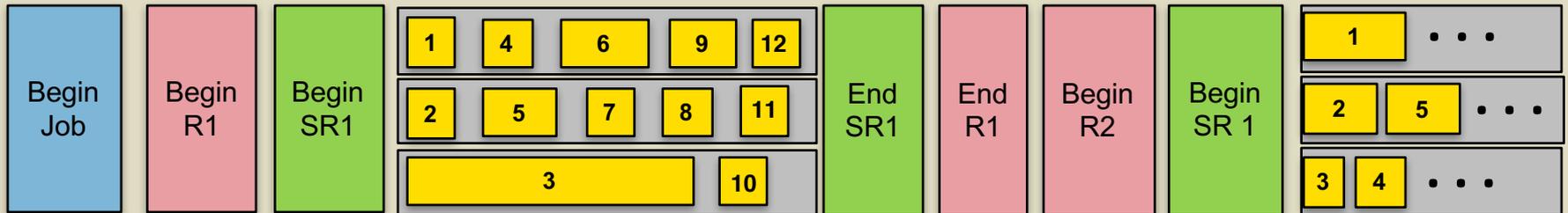
# art MT forum – Session 1

[https://cdcvs.fnal.gov/redmine/projects/art/wiki/Art\\_multi-threading\\_forum\\_-\\_introduction](https://cdcvs.fnal.gov/redmine/projects/art/wiki/Art_multi-threading_forum_-_introduction)

- We discussed motivations for a multi-threaded (MT) framework
- Largely based off of CMSSW’s design
  - We use Intel’s Threading Building Blocks (TBB)
  - Steps to be performed are factorized into *tasks*

You can think of a call to your module’s “produce” function as performing a task

## Implemented for *art* 3:



# art MT forum – Session 2

[https://cdcvs.fnal.gov/redmine/projects/art/wiki/Art\\_multi-threading\\_forum\\_-\\_modules](https://cdcvs.fnal.gov/redmine/projects/art/wiki/Art_multi-threading_forum_-_modules)

- *art* guarantees that any currently-existing modules (to within some interface changes) will be usable in a multi-threaded execution of *art*.
  - No multi-threading benefits will be realized with such “legacy” modules
- To take advantage of *art*'s multi-threading capabilities, users will need to choose the kind of module they use:
  - **Shared module**: sees all events—calls can be serialized or asynchronous.
  - **Replicated module**: for a configured module, one copy of that module is created per schedule—each module copy sees one event at a time. Use if moving to a concurrent, shared module is not feasible.

## art MT forum – Session 2

[https://cdcvs.fnal.gov/redmine/projects/art/wiki/Art\\_multi-threading\\_forum\\_-\\_modules](https://cdcvs.fnal.gov/redmine/projects/art/wiki/Art_multi-threading_forum_-_modules)

- Which kind of module you want is determined from the base class.
  - Producers that must be called serially due to a shared resource:

```
class Fitter : public art::shared::Producer {
public:
    explicit Fitter(Parameters const& p)
    {
        serialize<Event>("TFileService"); // Declare the common resource
    }

    // Called serially wrt. other modules that use TFileService
    void produce(Event& e) override {}
};
```

# art MT forum – Session 2

[https://cdcvs.fnal.gov/redmine/projects/art/wiki/Art\\_multi-threading\\_forum\\_-\\_modules](https://cdcvs.fnal.gov/redmine/projects/art/wiki/Art_multi-threading_forum_-_modules)

- Which kind of module you want is determined from the base class.
  - Producers that you guaranteed have no data races:

```
class HitMaker : public art::shared::Producer {
public:
    explicit HitMaker(Parameters const& p)
    {
        async<Event>();
    }

    void produce(Event& e) override {} // Called asynchronously
};
```

# art MT forum – Session 2

[https://cdcvs.fnal.gov/redmine/projects/art/wiki/Art\\_multi-threading\\_forum\\_-\\_modules](https://cdcvs.fnal.gov/redmine/projects/art/wiki/Art_multi-threading_forum_-_modules)

- Which kind of module you want is determined from the base class.

## – Replicated modules:

```
class Accumulator : public art::replicated::Producer {
public:
    explicit Accumulator(Parameters const& p)
    {}

    // Each module copy sees one event at a time
    void produce(Event& e) override;

    // Reduction interface...
};
```

## *art 3.0*

# General support

- Nominally all current facilities provided by *art* will be supported (some discussion later on).
- Current legacy modules will still be supported
- Shared and replicated modules will be supported
- Processing of concurrent events will be supported for producers and filters
- Concurrent processing of multiple trigger paths for the same event will be supported
- Concurrent insertion into an `art::Event` will be supported
- Analyzers and output modules will be serialized (for now)
- Services will need to be thread-safe

# Secondary input files

- Secondary input files (i.e. backing input files) are files that are opened whenever a data-product retrieval from the primary input file fails.
- They are specified in the RootInput source configuration.  
fileNames: ["f.root"]  
secondaryFileNames: [{ a: "f.root" b: ["g1.root", "g2.root"]}]
- Since these files can be opened at any time during event processing, they cause problems for multi-threading.

# Secondary input files

- Secondary input files (i.e. backing input files) are files that are opened whenever a data-product retrieval from the primary input file fails.
- They are specified in the RootInput source configuration.

```
fileNames: ["f.root"]  
secondaryFileNames: [{ a: "f.root" b: ["g1.root", "g2.root"]}]
```
- Since these files can be opened at any time during event processing, they cause problems for multi-threading.
- There are two options:
  - Since the current file-delivery system is not conducive to secondary input-file specification, it is unlikely that anyone is using it. **We can remove the facility altogether.** This eases the maintenance burden for us, but removes a feature that might be used later.
  - **Only allow secondary input files when 1 thread and 1 schedule are used.**
  - Thoughts?

# Product mixing

- Users can mix products into an event using the MixFilter template.
- Since this facility uses ROOT I/O, there are thread-safety issues.
- For that reason, **we propose that the event-level calls be serialized for mix filters until we can develop a more efficient solution.**

# Product mixing

- Users can mix products into an event using the MixFilter template.
- Since this facility uses ROOT I/O, there are thread-safety issues.
- For that reason, **we propose that the event-level calls be serialized for mix filters until we can develop a more efficient solution.**
- It may be possible for users to specify mixing operations that can be run in parallel for a given serialized event. **Is this of interest?**
- A caveat: with multi-threading enabled, repeated executions of the same program will likely not produce the same mixed events.

# TFileService constraints

- For modules to use it **safely**, they must register that it is a shared resource from within their constructors.

```
MyAnalyzer(Parameters const& p) {  
    serialize<Event>("TFileService");  
}
```

- With *art* 2.10, it is possible to enable TFileService file-switching by specifying the 'fileProperties' configuration table.
- File-switching is not thread-safe—**the facility can only be supported if 1 thread and 1 schedule are being used.**

# Data-product dependency checking

- It is a workflow error for a module on one path to require a product produced by a module on another path (e.g.):  
    p1: [a, b, c]  
    p2: [d, e]   // Error: 'd' uses a product created by 'c'
- The order in which paths are executed is unspecified—serious issue in multi-threading.
- *art* does not currently catch this type of error.

# Data-product dependency checking

- It is a workflow error for a module on one path to require a product produced by a module on another path (e.g.):

p1: [a, b, c]

p2: [d, e] // Error: 'd' uses a product created by 'c'

- The order in which paths are executed is unspecified—serious issue in multi-threading.
- *art* does not currently catch this type of error.
- In *art* 3.0, if you use the 'consumes' interface, we will detect the error and an exception will be thrown:
  - [https://cdcvs.fnal.gov/redmine/projects/art/wiki/Declaring\\_products\\_to\\_consume](https://cdcvs.fnal.gov/redmine/projects/art/wiki/Declaring_products_to_consume)

# Data-product dependency checking

- CMSSW has the concept of telling the system if a product should be retrieved from the current process or from the input source.
- We would like to adopt something similar to that:

```
InputTag const prodTag{"m1:i1:*current_process*"};  
InputTag const presTag{"m1:i2:*source*"};
```

- A user is not required to specify “\*current\_process\*” or “\*source\*”, but if he/she does:
  - Product lookup can be more efficient
  - If included in a consumes statement, potential ambiguities are resolved

# Random number generation

- The *art* implementation of RandomNumberGenerator (RNG) uses CLHEP.
- CLHEP random number engines have mutable state that pose problems for MT.
- *art* will provide an interface that, when used correctly, will ensure no data races.
- Data races might be avoidable entirely if users rely only on `createEngine` (which is a member of any module that would use RandomNumberGenerator).
- Calling `ServiceHandle<RNG>{}->getEngine(...)` may require a schedule ID to be passed as an argument.
- Random-number seeds are a problem for replicated modules:
  - A seed repeated across module copies results in correlated results
  - CMSSW solution is to take the seed specified by the configuration and increment by 1 for each copy. **Is this acceptable?**

# Replicated-module reduction facility

- A replicated module is replicated across schedules.
- After the events for a given SubRun have been processed, it may be desirable to combine data from across the schedules for (e.g.) writing to a histogram.
- **NB:** combining information for data products is unnecessary since *art*'s internal data-product mechanisms aggregate (Sub)Run products for you.
- Is such a reduction facility needed? If so, is it required for *art* 3.0?

## Next steps

- Some more implementation necessary.
- As we try to wrap up *art* 3.0, we will rely more heavily on experiment input.
- The goal is to be able to release by the end of March.
  - Will be difficult, but not crazy.
- We will be fleshing out the documentation.
- We plan to target specific experiment modules to demonstrate multi-threading capability.