



## *art news*

Kyle J. Knoepfel  
*art* stakeholders meeting  
11 May 2017



# Today's meeting

- Recent issues
- Preview of *art* 2.07
- Questions for the stakeholders
  - `MemoryTracker` database schema change
  - Behavior of calling `Post*` service callbacks
  - `art::DoNotRecordParents` base class
- GCC 7.1
- AOB

# Could not retrieve Assns in process that produced it

- A user reported not being able to retrieve an Assns in the same process that produced it.
- Upon investigation, this is what the code looked like:

```
MyProducer::MyProducer(Parameters const&)\n{\n    produces<Assns<A,B>>();\n}\n\nvoid MyProducer::produce(art::Event& e)\n{\n    auto ab = std::make_unique<Assns<B,A>>();\n    // ...\n    e.put(std::move(ab));\n}
```

# Could not retrieve Assns in process that produced it

- A user reported not being able to retrieve an Assns in the same process that produced it.
- Upon investigation, this is what the code looked like:

```
MyProducer::MyProducer(Parameters const&
{
    produces<Assns<A,B>>();
}

void MyProducer::produce(art::Event& e)
{
    auto ab = std::make_unique<Assns<B,A>>();
    // ...
    e.put(std::move(ab));
}
```

Reversed  
template arguments

# Could not retrieve Assns in process that produced it

- A user reported not being able to retrieve an Assns in the same process that produced it.
- Upon investigation, this is what the code looked like:

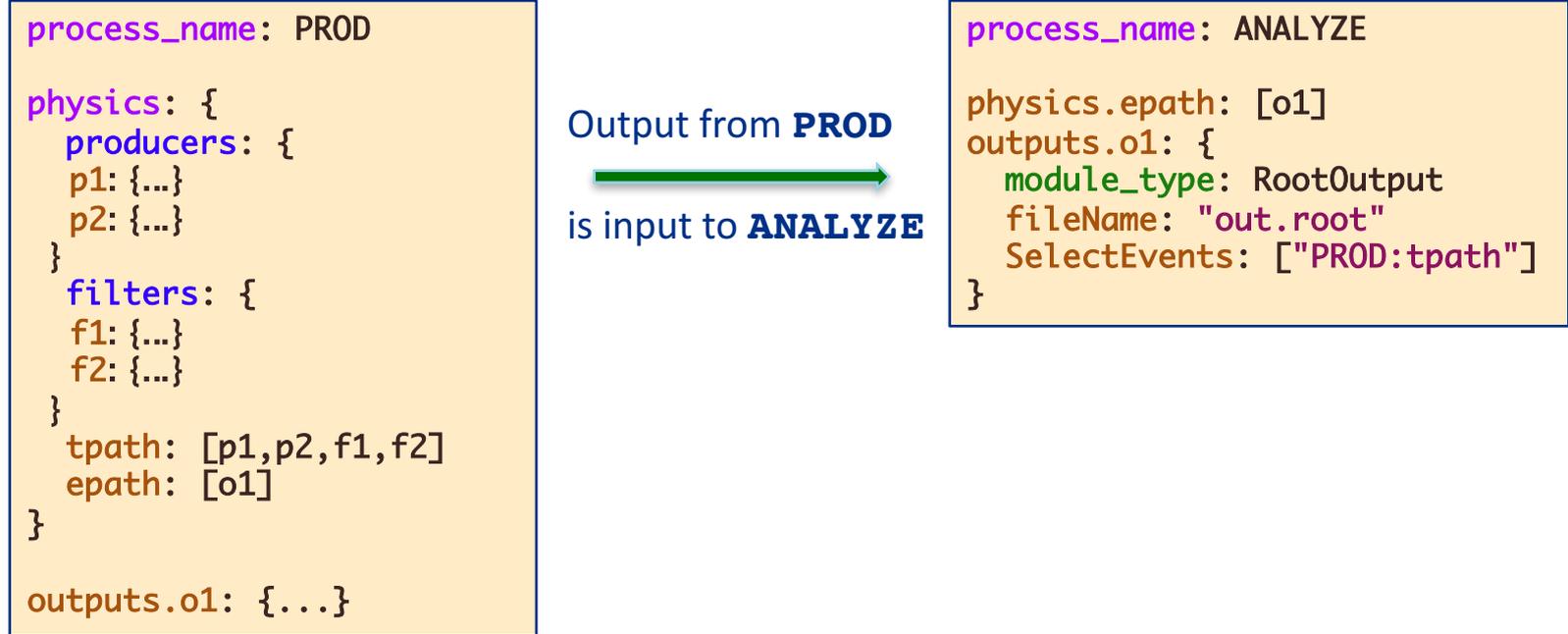
```
MyProducer::MyProducer(Parameters const&
{
    produces<Assns<A,B>>(C);
}

void MyProducer::produce(art::Event& e)
{
    auto ab = std::make_unique<Assns<B,A>>(C);
    // ...
    e.put(std::move(ab));
}
```

- *art* did not catch this bug because of insufficient checking of the types that were being inserted into the event.
- This has been fixed.

# SelectEvents according to filter criteria from previous process

- User asked about using a trigger path from a previous process in SelectEvents.
- Yes, that is intended to be supported. It would like this:



# SelectEvents according to filter criteria from previous process

- User asked about using a trigger path from a previous process in SelectEvents.
- Yes, that is intended to be supported. It would like this:

```
process_name: PROD

physics: {
  producers: {
    p1: {...}
    p2: {...}
  }
  filters: {
    f1: {...}
    f2: {...}
  }
  tpath: [p1,p2,f1,f2]
  epath: [o1]
}

outputs.o1: {...}
```

Output from **PROD**  
→  
is input to **ANALYZE**

```
process_name: ANALYZE

physics.ephath: [o1]
outputs.o1: {
  module_type: RootOutput
  fileName: "out.root"
  SelectEvents: ["PROD:tpath"]
}
```

- Specify process-qualified path  
    "<process name>:<trigger path>"
- Alas, does not work with *art* 2.06 (or any earlier version that you would care about).
- Fixed for *art* 2.07.

# Preview of *art* 2.07

- **General features:**

- **EventID comparator:** means of specifying using wildcard patterns which events you would like to match.
- FHiCL Python extension module, for converting FHiCL documents to Python dictionaries.
- `ServiceHandle<SomeService const>` constructions allowed
- `DatabaseConnection` service: provides ability to have thread-safe interactions with SQLite databases
- Improvements to `TimeTracker` reporting

- **Utilities in preparation for multi-threading:**

- **`cet::SimultaneousFunctionSpawner`:** class used for testing how a function behaves when multiple threads call it simultaneously.
- **`CET_ASSERT_ONLY_ONE_THREAD()`:** macro that is enabled when `NDEBUG` is false. It is used to ensure that only one thread is accessing the given block of code at a time. `std::abort` is called if a second thread accesses the block before the first one has finished. Same semantics as `assert(...)`.

# Preview of *art* 2.07

- **Under-the-cover improvements:**
  - All (relevant) registries and services are thread-safe
  - Message-logging via the message facility is thread-safe
- **Bug fixes:**
  - Restore graceful shutdown of *art* for exception throws and CTRL+C signals.
  - More robust checking for `Assns<A, B(, D)>` vs. `Assns<B, A(, D)>` insertions
  - Restore event-selection using previous-process filter criteria
  - etc.

# Preview of *art* 2.07

- **Breaking changes:**

- Module and service reconfiguration has been removed.
  - To our knowledge, no one except for `artdaq` was using this facility. `artdaq` no longer has need of reconfiguration.
  - `MyModule::reconfigure(ParameterSet const&)` functions are still allowed, but *art* does not implicitly call them.
  - `UserInteraction`, `SimpleInteraction` and `PathSelection` services have been removed.
- Functions of *art*-provided services that are intended to be used only as callbacks/implementation details are now private. Functions that are intended to be used via `ServiceHandle<SomeService>` are still available.
- `ServiceRegistry::instance()` is now private. Gaining access to a service can now be done only via `ServiceHandle<SomeService (const)>`.
- `MemoryTracker` database schema change.
- `floating_point_control` per-module configuration removed.

# MemoryTracker SQLite schema changes

- In order to provide thread-safe MemoryTracker services, the schema to the MemoryTracker database needed to be adjusted.

Run	SubRun	Event	Vsize	DeltaVsize	RSS	DeltaRSS
1	0	1	526.80859375	0	131.9453125	0



Step	Run	SubRun	Event	Vsize	RSS
PreProcessEvent	1	0	1	478.060544	137.306112
PostProcessEvent	1	0	1	478.060544	137.306112

- If a user wants to insert additional entries into an extant database but using the new schema, the insertion will fail. A migration tool will be necessary.
- **Is there a need to create a migration tool for current database files?**

# Potential change in behavior of services during an exception

- Whenever an event is processed, the `pre{Event,Module}` and `post{Event,Module}` callbacks are always called, even if one of the modules threw an exception during the call of its (e.g.) produce function.

# Potential change in behavior of services during an exception

- Whenever an event is processed, the `pre{Event,Module}` and `post{Event,Module}` callbacks are always called, even if one of the modules threw an exception during the call of its (e.g.) `produce` function.
- Such symmetry is manifested due to our use of the RAII idiom, which can be used to provide exception-safe code.

```
class ServiceSentry {
public:
    ServiceSentry() {
        preEventCallbacks().invoke();
    }
    ~ServiceSentry() noexcept(false) {
        postEventCallbacks().invoke();
    }
};
```

```
void processEvent(Event& e)
{
    ServiceSentry sentry;
    yourModule.produce(e);
}
```

# Potential change in behavior of services during an exception

- Whenever an event is processed, the `pre{Event,Module}` and `post{Event,Module}` callbacks are always called, even if one of the modules threw an exception during the call of its (e.g.) `produce` function.
- Such symmetry is manifested due to our use of the RAll idiom, which can be used to provide exception-safe code.

```
class ServiceSentry {  
public:  
    ServiceSentry() {  
        preEventCallbacks().invoke();  
    }  
    ~ServiceSentry() noexcept(false) {  
        postEventCallbacks().invoke();  
    }  
};  
  
void processEvent(Event& e)  
{  
    ServiceSentry sentry;  
    yourModule.produce(e);  
}
```

## processEvent call sequence

1. `preEventCallbacks().invoke();`
2. `yourModule.produce(e);`
3. `postEventCallbacks().invoke();`

# Potential change in behavior of services during an exception

- Whenever an event is processed, the `pre{Event,Module}` and `post{Event,Module}` callbacks are always called, even if one of the modules threw an exception during the call of its (e.g.) `produce` function.
- Such symmetry is manifested due to our use of the RAII idiom, which can be used to provide exception-safe code.

```
class ServiceSentry {  
public:  
    ServiceSentry() {  
        preEventCallbacks().invoke();  
    }  
    ~ServiceSentry() noexcept(false) {  
        postEventCallbacks().invoke();  
    }  
};
```

```
void processEvent(Event& e)  
{  
    ServiceSentry sentry;  
    yourModule.produce(e);  
}
```

## processEvent call sequence

1. `preEventCallbacks().invoke();`
2. `yourModule.produce(e);`
3. `postEventCallbacks().invoke();`

If `produce` throws  
and then one of the  
post event callbacks  
throws →  
**std::terminate**

# Potential change in behavior of services during an exception

- Various solutions to this problem.
- In multi-threaded *art*, we will probably need to adopt a couple of them.
- Our suggestion for this scenario is to remove the sentry all together and make explicit calls to the pre/post callbacks:

# Potential change in behavior of services during an exception

- Various solutions to this problem.
- In multi-threaded *art*, we will probably need to adopt a couple of them.
- Our suggestion for this scenario is to remove the sentry all together and make explicit calls to the pre/post callbacks:

```
class ServiceSentry {
public:
    ServiceSentry() {
        preEventCallbacks().invoke();
    }
    ~ServiceSentry() noexcept(false) {
        postEventCallbacks().invoke();
    }
};

void processEvent(Event& e)
{
    ServiceSentry sentry;
    yourModule.produce(e);
}
```



```
void processEvent(Event& e)
{
    preEventCallbacks().invoke();
    yourModule.produce(e);
    postEventCallbacks().invoke();
}
```

# Potential change in behavior of services during an exception

- Semantics of Post\* callbacks changes: (e.g.) post-event callbacks are invoked whenever the process-event function calls do not result in an exception throw.
- A consequence:
  - For example, the `TimeTracker` will not record the execution time for a module whose produce threw an exception.
- **Is this change in behavior acceptable to the experiments?**

## **art::DoNotRecordParents base class**

- *art* tracks the products that were retrieved (parents) during a `produce/filter` call.
- For any product that is placed onto the event, the parents are associated with it in the metadata.
- It is possible to tell *art* to NOT track the parents by making your product inherit from the `art::DoNotRecordParents` base class.
- Based on our trolling of experiment/project repositories, no one is using this facility.
- Removing the facility would simplify our code.
- No immediate need to remove it, but something for the experiments to consider.

# GCC 7.1

- Released 5/2/2017
- C++17 Draft International Standard released for ISO ballot in March.
  - No major changes expected before final approval this summer
- GCC 7.1 is C++17 (core language) feature complete
- At some point in the near future, we will want to move toward GCC 7.1, and also want to start compiling the `c++1z` flag.
- We may also want to start using the Concepts Lite features (e.g. constrained templates) that are slated to become part of C++20.

Language Feature	Proposal	Available in GCC?	SD-6 Feature Test
Removing trigraphs	N4086	5.1	
u8 character literals	N4267	6	cpp_unicode_characters >= 201411
Folding expressions	N4295	6	cpp_fold_expressions >= 201411
Attributes for namespaces and enumerators	N4266	4.9 (namespaces) 6 (enumerators)	cpp_namespace_attributes >= 201411 cpp_enumerator_attributes >= 201411
Nested namespace definitions	N4230	6	cpp_nested_namespace_definitions >= 201411
Allow constant evaluation for all non-type template arguments	N4268	6	cpp_nontype_template_args >= 201411
Extending static_assert	N3928	6	cpp_static_assert >= 201411
New Rules for auto deduction from braced-init-list	N3922	5	
Allow typename in a template template parameter	N4051	5	
[[fallthrough]] attribute	P0188R1	7	has_cpp_attribute(fallthrough)
[[nodiscard]] attribute	P0189R1	4.8 ([[gnu::warn_unused_result]]) 7 (P0189R1)	has_cpp_attribute(nodiscard)
[[maybe_unused]] attribute	P0212R1	4.8 ([[gnu::unused]]) 7 (P0212R1)	has_cpp_attribute(maybe_unused)
Extension to aggregate initialization	P0017R1	7	cpp_aggregate_bases >= 201603
Wording for constexpr lambda	P0170R1	7	cpp_constexpr >= 201603
Unary Folds and Empty Parameter Packs	P0036R0	6	cpp_fold_expressions >= 201603
Generalizing the Range-Based For Loop	P0184R0	6	cpp_range_based_for >= 201603
Lambda capture of *this by Value	P0018R3	7	cpp_capture_star_this >= 201603
Construction Rules for enum class variables	P0138R2	7	
Hexadecimal floating literals for C++	P0245R1	3.0	cpp_hex_float >= 201603
Dynamic memory allocation for over-aligned data	P0035R4	7	cpp_aligned_new >= 201606
Guaranteed copy elision	P0135R1	7	
Refining Expression Evaluation Order for Idiomatic C++ constexpr if	P0145R3 P0292R2	7 7	cpp_if_constexpr >= 201606
Selection statements with initializer	P0305R1	7	
Template argument deduction for class templates	P0091R3	7	cpp_deduction_guides >= 201606
Declaring non-type template parameters with auto	P0127R2	7	cpp_template_auto >= 201606
Using attribute namespaces without repetition	P0028R4	7	
Ignoring unsupported non-standard attributes	P0283R2	Yes	
Structured bindings	P0217R3	7	cpp_structured_bindings >= 201606
Remove Deprecated Use of the register Keyword	P0001R1	7	
Remove Deprecated operator++(bool)	P0002R1	7	
Make exception specifications be part of the type system	P0012R1	7	cpp_noexcept_function_type >= 201510
_has_include for C++17	P0061R1	5	
Rewording inheriting constructors (core issue 1941 et al)	P0136R1	7	cpp_inheriting_constructors >= 201511
Inline variables	P0386R2	7	cpp_inline_variables >= 201606
DR 150, Matching of template template arguments	P0522R0	7	cpp_template_template_args >= 201611
Removing dynamic exception specifications	P0003R5	7	
Pack expansions in using-declarations	P0195R2	7	cpp_variadic_using >= 201611
A byte type definition	P0298R0	7	