



art news

Kyle J. Knoepfel
art stakeholders meeting
9 February 2017



art 2.06

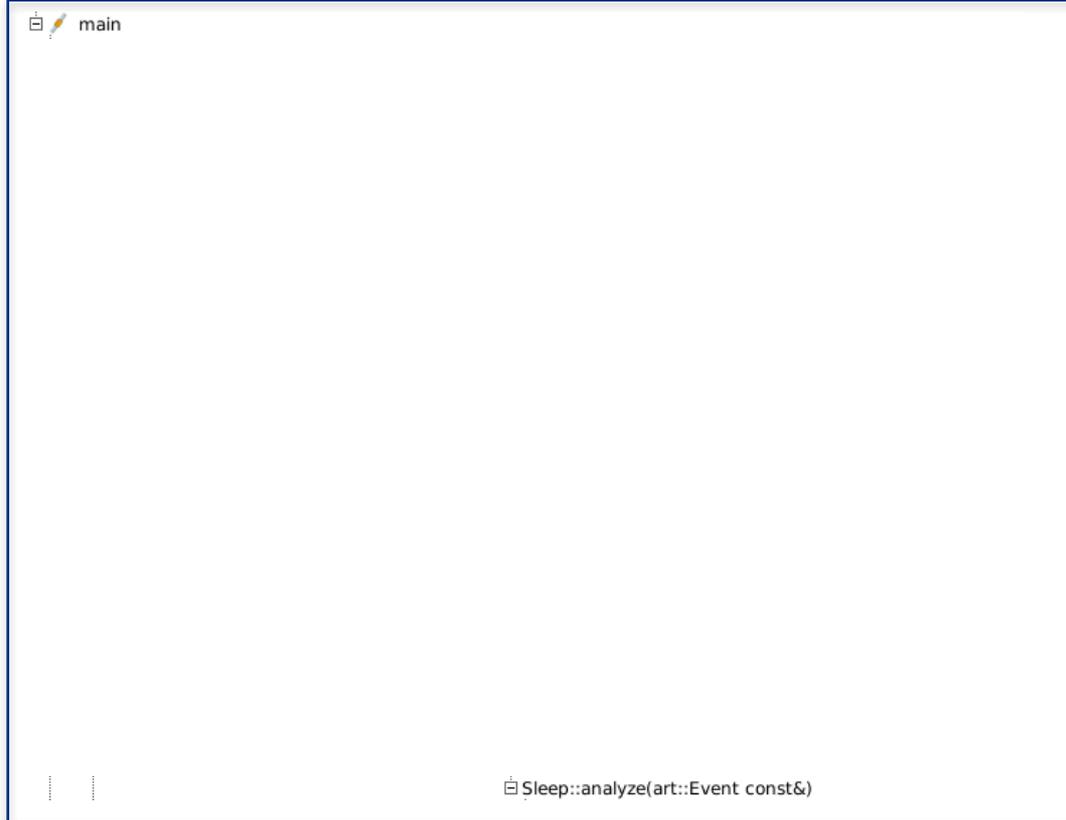
- New Platforms:
 - Ubuntu 16.04 LTS
 - Ubuntu 14.04 LTS may be built on request for 2.06.01 and later; will be dropped from *art* 2.07
 - macOS Sierra (Darwin 16), SIP disabled
- Notable external product changes:
 - ROOT 6.08.04c
 - GCC 6.3 (e14)
- New features:
 - Extended **MixFilter** abilities for `Events` and their associated `(Sub)Run` products
 - Relaxed `art::Assns<A,B>` lookup policy and relaxed `art::Ptr<T>` resolution rules for smart query objects
 - Introduction of the *art* “tool”
- Some breaking changes

art 2.06.00 vs. 2.06.01

- *art* 2.06.01 supports the e10 qualifier (GCC 4.9.3a); 2.06.00 does not.
- *art* 2.06.01 fixes a conditional compilation error that could occur during interactive ROOT sessions that rely on the utilities in `cetlib/container_algorithms.h`.

https://cdcvs.fnal.gov/redmine/projects/art/wiki/Release_Notes_20600

Changes with the event loop



```
main
...
Sleep::analyze(art::Event const&)
```

- From the top-level `main` function, down to event-level module overrides, there is quite a bit of intervening code.
- To illustrate this, I created an analyzer called `Sleep` that sleeps for 1 second per event.
- At the left is the stack trace from `main` to `Sleep::analyze`.

Changes with the event loop

main

```
art -c ...
```

Your analyze override

```
Sleep::analyze(art::Event const&)
```

- From the top-level `main` function, down to event-level module overrides, there is quite a bit of intervening code.
- To illustrate this, I created an analyzer called `Sleep` that sleeps for 1 second per event.
- At the left is the stack trace from `main` to `Sleep::analyze`.

Changes with the event loop

```
main
├─ artapp(int, char**)
│  └─ art::run_art(int, char**, boost::program_options::options_description&, cet::filepath_maker&, std::vector<...
│     └─ art::run_art_common(fhicl::ParameterSet const&, art::detail::DebugOutput)
│        └─ art::EventProcessor::runToCompletion()
│           └─ process_event [inlined]
│              └─ boost::statechart::state_machine<statemachine::Machine, statemachine::Starting, std::allocator<v...
│                 └─ boost::statechart::state_machine<statemachine::Machine, statemachine::Starting, std::allocator<...
│                    └─ operator()<boost::statechart::detail::send_function<boost::statechart::detail::state_base<std::al...
│                       └─ operator() [inlined]
│                          └─ boost::statechart::simple_state<statemachine::NewEvent, statemachine::HandleEvents, boo...
│                             └─ local_react<boost::mpl::list<boost::statechart::transition<statemachine::Process, statemac...
│                                └─ local_react_impl<boost::mpl::list<boost::statechart::transition<statemachine::Process, st...
│                                   └─ react<statemachine::NewEvent, boost::statechart::event_base, void const*>
│                                      └─ react
│                                         └─ react
│                                            └─ react
│                                               └─ react_without_action
│                                                  └─ transit<statemachine::ProcessEvent>
│                                                     └─ transit_impl<statemachine::ProcessEvent, statemachine::Machine, boost::stat...
│                                                        └─ construct
│                                                           └─ deep_construct
│                                                              └─ shallow_construct [inlined]
│                                                                 └─ statemachine::ProcessEvent::ProcessEvent(boost::statechart::state<sta...
│                                                                    └─ art::EventProcessor::processEvent()
│                                                                       └─ art::EventProcessor::process_<art::ProcessPackage<(art::Level)0
│                                                                          └─ art::EndPathExecutor::process<art::ProcessPackage<(art::Level)0
│                                                                             └─ art::Path::process<art::ProcessPackage<(art::Level)0
│                                                                                └─ runWorker<art::ProcessPackage<(art::Level)0> > [inlined]
│                                                                                   └─ art::Worker::doWork<art::ProcessPackage<(art::Level)0
│                                                                                      └─ invoke<art::EventPrincipal> [inlined]
│                                                                                         └─ art::WorkerT<art::EDAnalyzer>::implDoProcess(art::Event...
│                                                                                            └─ art::EDAnalyzer::doEvent(art::EventPrincipal const&, cet...
│                                                                                               └─ Sleep::analyze(art::Event const&)
```

- From the top-level `main` function, down to event-level module overrides, there is quite a bit of intervening code.
- To illustrate this, I created an analyzer called `Sleep` that sleeps for 1 second per event.
- At the left is the stack trace from `main` to `Sleep::analyze`.

With state machine

```
main
├─ artapp(int, char**)
│  └─ art::run_art(int, char**, boost::program_options::options_description&, cet::filepath_maker&, std::vector<...
│     └─ art::run_art_common(fhicl::ParameterSet const&, art::detail::DebugOutput)
│        └─ art::EventProcessor::runToCompletion()
│           └─ process_event [inlined]
│              └─ boost::statechart::state_machine<statemachine::Machine, statemachine::Starting, std::allocator<v...
│                 └─ boost::statechart::state_machine<statemachine::Machine, statemachine::Starting, std::allocator<...
│                    └─ operator() [inlined]
│                       └─ boost::statechart::simple_state<statemachine::NewEvent, statemachine::HandleEvents, boo...
│                          └─ local_react<boost::mpl::list<boost::statechart::transition<statemachine::Process, statemac...
│                             └─ local_react_impl<boost::mpl::list<boost::statechart::transition<statemachine::Process, st...
│                                └─ react<statemachine::NewEvent, boost::statechart::event_base, void const*>
│                                   └─ react
│                                      └─ react
│                                         └─ react
│                                            └─ react_without_action
│                                               └─ transit<statemachine::ProcessEvent>
│                                                  └─ transit_impl<statemachine::ProcessEvent, statemachine::Machine, boost::stat...
│                                                     └─ construct
│                                                        └─ deep_construct
│                                                           └─ shallow_construct [inlined]
│                                                              └─ statemachine::ProcessEvent::ProcessEvent(boost::statechart::state<sta...
│                                                                 └─ art::EventProcessor::processEvent()
│                                                                    └─ art::EventProcessor::process_<art::ProcessPackage<(art::Level)0
│                                                                       └─ art::EndPathExecutor::process<art::ProcessPackage<(art::Level)0
│                                                                          └─ art::Path::process<art::ProcessPackage<(art::Level)0
│                                                                             └─ runWorker<art::ProcessPackage<(art::Level)0> > [inlined]
│                                                                                └─ art::Worker::doWork<art::ProcessPackage<(art::Level)0
│                                       └─ invoke<art::EventPrincipal> [inlined]
│                                          └─ art::WorkerT<art::EDAnalyzer>::implDoProcess(art::Event...
│                                             └─ art::EDAnalyzer::doEvent(art::EventPrincipal const&, cet...
│                                                └─ Sleep::analyze(art::Event const&)
```

- The red-highlighted section is the stack trace contribution due to using the Boost Statechart library to implement *art*'s state machine.
- In anticipation of complications that can arise with multi-threading, we decided to remove the state machine and use a set of nested for loops to express the allowed transitions.

Without state machine

```
main
├─ artapp(int, char**)
│  └─ art::run_art(int, char**, boost::program_options::options_description&, cet::filepath_maker&, std::v...
│     └─ art::run_art_common(fhicl::ParameterSet const&, art::detail::DebugOutput)
│        └─ art::EventProcessor::runToCompletion()
│           └─ art::EventProcessor::process<(art::Level)0>()
│              └─ process<(art::Level)1>
│                 └─ process<(art::Level)2>
│                    └─ process<(art::Level)3> [inlined]
│                       └─ art::EventProcessor::process<(art::Level)4>()
│                          └─ art::EventProcessor::processEvent()
│                             └─ art::EventProcessor::process_<art::ProcessPackage<(art::Level)4
│                                └─ art::EndPathExecutor::process<art::ProcessPackage<(art::Level)4
│                                   └─ art::Path::process<art::ProcessPackage<(art::Level)4
│                                      └─ runWorker<art::ProcessPackage<(art::Level)4 > > [inlined]
│                                         └─ art::Worker::doWork<art::ProcessPackage<(art::Level)4
│                                            └─ invoke<art::EventPrincipal> [inlined]
│                                               └─ art::WorkerT<art::EDAnalyzer>::implDoProcess(art::EventPrincipal&, art::C...
│                                                  └─ art::EDAnalyzer::doEvent(art::EventPrincipal const&, cet::exempt_ptr<a...
│                                                     └─ Sleep::analyze(art::Event const&)
```

- The red-highlighted section is the stack trace contribution due to the nested for loops.
- The code is now:
 - easier to understand
 - better suited for processing concurrent Events, SubRuns, and Runs
- Unlikely users would notice any difference in processing time.