Table of Contents

Writing a Thread-Friendly EDAnalyzer for Analysis

Complete:

Contacts

- **Contacts**: Christopher Jones (FNAL),
- **Hypernews forum**: https://hypernews.cern.ch/HyperNews/CMS/get/edmFramework.html
- **Documentation contact**: Christopher Jones

Thread Safety Requirements

Modules which are thread-friendly can only call code which either

- never reads or writes to memory data that can be written to by another module on another thread or
- reads or writes to memory which was properly protected from multi-threads.

This primarily reduces to not calling code which uses non-const globals or statics. The biggest worry is using classes from ROOT since behind the scene ROOT often uses globals. The following items are safe to call from an EDAnalyzer.

- Using a `TFile`, `TTree` and `TBranch`
- Using a histogram
  - except calling `Fit(const char*, ...)` is not safe. Instead pass a `TF1` as the first argument.
- Using TMVA reader (writing is not guaranteed to be safe)

The following items are known to be unsafe

- `TFormula::SetName`
- `TH1::Fit(const char*, ...)`

Legacy `EDAnalyzer`

A Legacy EDAnalyzer is one that inherits from the base class `edm::EDAnalyzer` which was used for the single-threaded framework:

```
#include "FWCore/Framework/interface/EDAnalyzer.h"
//use an anonymous namespace since there is no need for any other file to see these class names
namespace {
  class ExampleAnalyzer : public edm::EDAnalyzer {
   public:
      ExampleAnalyzer(edm::ParameterSet const& iPSet) {}

      void analyze(edm::Event const& iEvent, edm::EventSetup const&) override {}
   };
}
```

Such `EDAnalyzers` will run properly in the multi-threaded framework, however, they will not run efficiently. The multi-threaded framework must assume that all legacy modules are thread unsafe which includes assuming that any two legacy modules might be sharing the same memory (e.g. via a global). Therefore the framework will only run one legacy module at a time. So if two or more threads are at the point where the only modules available to run at a given time are all legacy modules, only one thread will actually do work while the other threads wait their turn. This wastes CPU time.

Thread-friendly `EDAnalyzer`

An `EDAnalyzer` which satisfies the thread safety requirements mentioned earlier can be converted into an `one` module. A `one` module is a module type which the multi-threaded framework assumes does not share memory with any other module and which wants to see every event in the job but can only handle seeing one event at a time (as apposed to many events simultaneously).

A `one` module allows the framework to run other modules at the same time it is running an instance of a `one` modules. However, if the only work the framework has to two on two different threads (each processing a differnt Event) is to run a particular `one` module, then only one of the thread will run while the other waits. In this case we are still wasting some CPU time, however it is much less common than for the legacy module case.

The full documentation about `one` modules chan be found here.

simple `one` module

It is easy to convert an `EDAnalyzer` which only cares about Events and not Runs or LuminosityBlocks into a `one` module. All one has to do is inherit from a new base class: `edm::one::EDAnalyzer<>`. The member functions remain the same.

```
#include "FWCore/Framework/interface/one/EDAnalyzer.h"
//use an anonymous namespace since there is no need for any other file to see these class names
namespace {
  class ExampleAnalyzer : public edm::one::EDAnalyzer<> {
   public:
      ExampleAnalyzer(edm::ParameterSet const& iPSet) {}

      void beginJob() override {}
      void analyze(edm::Event const& iEvent, edm::EventSetup const&) override {}
      void endJob() override {}
   };
}
```

watching for Run and transitions

If an `EDAnalyzer` needs to be told about a Run or LuminosityBlock transition (e.g. begin Run) than one must explicitly inform the framework about that requirement. This is done by passing a particular template argument to the base class. The reason the framework needs to be informed is if a `one` module needs to see these transitions than the framework will only be allowed to process one Run or one LuminosityBlock at a time. Such a restriction can once again reduce CPU efficiency.

watching for Runs

```
#include "FWCore/Framework/interface/one/EDAnalyzer.h"
//use an anonymous namespace since there is no need for any other file to see these class names
namespace {
  class ExampleAnalyzer : public edm::one::EDAnalyzer<edm::one::WatchRuns> {
   public:
      ExampleAnalyzer(edm::ParameterSet const& iPSet) {}

      void beginJob() override {}
      void beginRun(edm::Run const& iEvent, edm::EventSetup const&) override {}
      void analyze(edm::Event const& iEvent, edm::EventSetup const&) override {}
      void endRun(edm::Run const& iEvent, edm::EventSetup const&) override {}
      void endJob() override {}
   };
}
```

watching for

```
#include "FWCore/Framework/interface/one/EDAnalyzer.h"
//use an anonymous namespace since there is no need for any other file to see these class names
namespace {
  class ExampleAnalyzer : public edm::one::EDAnalyzer<edm::one::WatchLuminosityBlocks> {
   public:
      ExampleAnalyzer(edm::ParameterSet const& iPSet) {}

      void beginJob() override {}
      void beginLuminosityBlock(edm::LuminosityBlock const& iEvent, edm::EventSetup const&) overr
      void analyze(edm::Event const& iEvent, edm::EventSetup const&) override {}
      void endLuminosityBlock(edm::LuminosityBlock const& iEvent, edm::EventSetup const&) overrid
      void endJob() override {}
   };
}
```

watching for Both

```
#include "FWCore/Framework/interface/one/EDAnalyzer.h"
//use an anonymous namespace since there is no need for any other file to see these class names
namespace {
  class ExampleAnalyzer : public edm::one::EDAnalyzer<edm::one::WatchLuminosityBlocks,edm::one::W
   public:
      ExampleAnalyzer(edm::ParameterSet const& iPSet) {}

      void beginJob() override {}
      void beginRun(edm::Run const& iEvent, edm::EventSetup const&) override {}
      void beginLuminosityBlock(edm::LuminosityBlock const& iEvent, edm::EventSetup const&) overr
      void analyze(edm::Event const& iEvent, edm::EventSetup const&) override {}
      void endLuminosityBlock(edm::LuminosityBlock const& iEvent, edm::EventSetup const&) overrid
      void endRun(edm::Run const& iEvent, edm::EventSetup const&) override {}
      void endJob() override {}
   };
}
```

Using

The `TFileService` is not thread-safe since it is sharing the same memory (from the `TFile`) between different threads. It is, however, possible to safely access the `TFileService` in a `one` module by declaring that the module depends upon the `TFileService`. This declaration is done by using the *shared resources* mechanism. The *shared resources* mechanism lets the framework know that multiple modules are using the same non-thread safe resource, in this case the `TFileService` and that only one such sharing module should be run at a time. Other modules which are not using that shared resource are allowed to run concurrently with the one module which does use that resource.

```
#include "FWCore/Framework/interface/one/EDAnalyzer.h"
//use an anonymous namespace since there is no need for any other file to see these class names
namespace {
  class ExampleAnalyzer : public edm::one::EDAnalyzer<edm::one::SharedResources> {
   public:
      ExampleAnalyzer(edm::ParameterSet const& iPSet) {
          //must state that we are using the TFileService
          usesResource("TFileService");
      }

      void beginJob() override {}
      void analyze(edm::Event const& iEvent, edm::EventSetup const&) override {}
      void endJob() override {}
   };
}
```

Review status

Reviewer/Editor and Date	Comments
Last reviewed by: ChrisDJones - 13-Nov-2012	Created the page

Responsible: ChrisDJones
Last reviewed by: