Table of Contents

4.5 Classes for Event Selection

Complete:
Detailed Review status

Goals of this page:

This page is intended to familiarize you with the EventSelectorBase class and the classes derived from it.

In particular, this page demonstrates how to use those classes:

- within an EDAnalyzer
- stand-alone as EDFilters

Contents

#Motivation

Motivation

The Idea behind introducing a common base class for event selections is to use code in the

- CMS.PhysicsTools/RecoAlgos
- CMS.PhysicsTools/UtilAlgos

packages as EDFilters (this is what the code has been used for in the past) and to use the same code also within EDAnalyzers or in FWLite)

Event Selection Base Class

As base class for event selections, the class EventSelectorBase* (in the CMS.PhysicsTools/UtilAlgos package) is defined:

```
{
  class EventSelectorBase
  {
   public:
    // constructor
    explicit EventSelectorBase() {}

    // destructor
    virtual ~EventSelectorBase() {}

    // function implementing actual cut
    virtual bool operator()(edm::Event&, const edm::EventSetup&) = 0;
};
```

The return value of the operator() is true (false) in case the event given as argument of the operator passes (fails) the event selection.

Note that EventSelectorBase is a pure virtual class. Its purpose is to define the signature of the overloaded operator() that is to be implemented by all derived classes.

The overloaded operator() takes two arguments: a reference to an edm::Event plus a reference to the edm::EventSetup. In most cases, the reference to the edm::EventSetup will not be needed by the implementation of most derived classes. The edm::EventSetup has been added to the signature in order to include special cases which do need access to the edm::EventSetup and in order to resemble the full EDFilter interface.

Derived Classes

Classes derived fromEventSelectorBase need to implement the overloaded operator().

An example for such implementation is the class ObjectCountEventSelector (an adaption of the former ObjectCountFilter class to the interface defined by EventSelectorBase) in theCMS.PhysicsTools/UtilAlgos package:

```
template<typename C,
         typename S = AnySelector,
         typename N = MinNumberSelector,
         typename CS = typename helper::CollectionFilterTrait<C, S, N>::type>
class ObjectCountEventSelector : public EventSelectorBase
{
 public:
  /// constructor
  explicit ObjectCountEventSelector( const edm::ParameterSet & cfg ) :
    src_( cfg.template getParameter<edm::InputTag>( "src" ) ),
    select_( reco::modules::make<S>( cfg ) ),
    sizeSelect_( reco::modules::make<N>( cfg ) ) {
  }

  bool operator()(edm::Event& evt, const edm::EventSetup&) {
    edm::Handle<C> source;
    evt.getByLabel( src_, source );
    return CS::filter( * source, select_, sizeSelect_ );
  }

 private:
  /// source collection label
  edm::InputTag src_;

  /// object filter
  S select_;

  /// minimum number of entries in a collection
  N sizeSelect_;
};
```

The ObjectCountEventSelector class in this example allows to select events with a certain number of physics objects (particles, tracks, clusters...)

Usage of derived Classes within an EDAnalyzer

In order to use a class derived from EventSelectorBase in an EDAnalyzer, you have two options: Either

simply create on object of the derived class explicitely:

```
EventSelectorBase* eventSelector = new MyEventSelector();
...
if ( (*eventSelector)(evt, es) ) { // evt (es) is a reference to an edm::Event (edm::EventSetup
  // fill histograms
}
```

or create an object via the plugin mechanism of the EDM framework:

```
EventSelectorBase* eventSelector = EventSelectorPluginFactory::get()->create("MyEventSelector",
...
if ( (*eventSelector)(evt, es) ) {
  // fill histograms
}
```

Which of the two methods you use for creating objects in your EDAnalyzer is your own choice.

Note that due the fact that derived class objects inherit from EventSelectorBase and due to the fact that operator() is a virtual function, you can store the event selector objects in arrays and write generic code to check if an event either passes or fails to pass all selections:

```
std::vector<EventSelectorBase*> eventSelectors;
eventSelector.push_back(new MyEventSelector1());
eventSelector.push_back(new MyEventSelector2());
...
bool passesEventSelection = true;
for (std::vector<EventSelectorBase*>::iterator it = eventSelectors.begin();
     it != eventSelectors.end(); ++it ) {
   if ( !(**it)(evt,es) ) passesEventSelection = false;
}

if ( passesEventSelection ) {
  // fill histograms
}
```

Usage of derived Classes as EDFilters

All classes derived from EventSelectorBase can be declared to be EDFilters by means of a simple typedef:

```
template<typename C,
         typename S = AnySelector,
         typename N = MinNumberSelector,
         typename CS = typename helper::CollectionFilterTrait<C, S, N>::type>
struct ObjectCountFilter {
  typedef EventSelectorAdapter< ObjectCountEventSelector<C, S, N, CS> > type;
};
```

The class EventSelectorAdapter used in this typedef provides the interface between the overloaded operator() of the EventSelectorBase class and the filter() method required for a class to be an EDFilter.

(In case you are wondering why ObjectCountFilter has been defined as a typedef within a struct: the nesting of a typedef within a struct is a technical necessity, resulting from the fact that the C++ standard does not support template typedefs directly - see e.g. the following [[http://www.gotw.ca/gotw/079.htm[article by Herb Sutter]].)

Future Work

In case you would use classes other then ObjectCountEventSelector for event selection within your EDAnalyzer or in FWLite which exists as EDFilters e.g. in the CMS.PhysicsTools/RecoAlgos or CMS.PhysicsTools/UtilAlgos packages, you would need to:

- open the original .h and .cc files (e.g. ObjectCountFilter.h)
- modify the existing code to inherit from EventSelectorBase instead of from EDFilter
- save the modified .h and .cc files under a new name (replace 'Filter' by 'EventSelector'; e.g. 'ObjectCountEventSelector.h')
- create a new .h file in which define an EDFilter by a typedef as explained in section Usage of derived Classes as EDFilters
- save the new .h file under the original name of the EDFilter source file (e.g. ObjectCountFilter.h)
- *In case the class which you modified contains template parameters **and** the templated class is used in typedefs, you will need to add '::type' to each typedef, e.g. you would need to change:*

```
typedef ObjectCountFilter<std::vector<Muon>, AnySelector, MinNumberSelector> PATMuonMinFilter;
```

to

```
typedef ObjectCountFilter<std::vector<Muon>, AnySelector, MinNumberSelector>::type PATMuonMinFi
```

(example taken from CMS.PhysicsTools/PatAlgos/PATObjectFilter.h)

Review status

Reviewer/Editor and Date	Comments
Main.Christian Veelken - 16 Dec 2008	Initial Version

Responsible: Main.ChristianVeelken
Last reviewed by: Main.BenediktHegner - 19 Dec 2008

This topic: CMSPublic > SWGuideEventSelector
Topic revision: r3 - 2009-01-27 - JochenOtt