Table of Contents

4.3 Particle Candidates Utilities and Framework Modules

Complete:
Detailed Review status

Goals of this page:

This page is intended to familiarize you with the common set of classes and tools which are used to develop a modular Physics Analysis software using the Framework and Event Data Model.

Contents

Candidate Framework Modules

Generic framework modules to manipulate Particle Candidates are provided. In particular:

Candidate Selectors

The module `CandSelector` selects Particle Candidates with specified cuts that can be specified by the user via a configurable string. Example of the configuration is:

```
process.goodMuons = cms.EDFilter("CandSelector",
    src = cms.InputTag("selectedLayer1Muons"),
    cut = cms.string("pt > 5.0")
)
```

This will take the PAT muons as described here and select those which have a transverse momentum larger than 5 GeV/c.

More details on available candidate selectors can be found in:

- Candidate Selector SW Guide

Candidate Combiners

Combiner modules compose other particles to create `CompositeCandidate`. daughter particles kinematics is copied into the composite object, and links to the original "master" particles can be stored using `ShallowCloneCandidate`.

An example of usage of such modules is the following:

```
process.zToMuMu = cms.EDProducer("CandViewShallowCloneCombiner",
    decay = cms.string('selectedLayer1Muons@+ selectedLayer1Muons@-'),
    cut = cms.string('50 < mass < 120'),
)
```

This will take the PAT layer 1 muons that have opposite sign, and combine them into `Z` candidates, throwing away those candidates outside of the mass range from 50 to 120 GeV/c2.

More details on available candidate selectors can be found in the following document:

- Combinatorial Analysis tools using particle candidates.

It is also possible to specify an optional name and daughter roles for these tools, like:

```
process.zToMuMu = cms.EDProducer("CandViewShallowCloneCombiner",
    decay = cms.string('selectedLayer1Muons@+ selectedLayer1Muons@-'),
    cut = cms.string('50.0 < mass < 120.0'),
    name = cms.string('zToMuMu'),
    roles = cms.vstring('muon1', 'muon2')
)
```

This will automatically assign names and roles as described here. The rest of the functionality is the same as the previous example.

Other Modules

A more complete list of the available modules to manipulate collections of candidates can be found in:

- Candidate Common Modules SW Guide

Candidate Utilities

Utilities are provided to perform the most common operations on Particle Candidates.

Overlap Checking

Overlap between two candidates occurs if the two candidates, or any of their daughters, share one of the components (a track, a super-cluster, etc.). The utility `OverlapChecker` checks for overlap occurrences:

```
#include "DataFormats/Candidate/interface/OverlapChecker.h"

OverlapChecker overlap;
const Candidate & c1 = ..., & c2 = ...;
if (overlap( c1, c2 ) )  { ... }
```

Note: this overlap checking mechanism only looks for identical components in the candidate decay chain, but has no way to check, for instance, of two candidates are made from tracks or clusters that share a common set of hits. More advanced utilities should be used for such more refined overlap checking.

Candidate Setup Utilities

"Setup" utilities allow to modify the candidate content (momentum, vertex, ...). The simplest provided setup utility is:

- `AddFourMomenta`: sets a candidate's 4-momentum adding 4-momenta of its daughters. In the following example, a Composite Candidate is created, its two daughters are added to it, and its momentum is set as the sum of the daughters four-momenta:

```
CompositeCandidate comp;
comp.addDaughter( dau1 );
comp.addDaughter( dau2 );
AddFourMomenta addP4;
addP4.set( comp );
```

Boosting Candidates

If you want for to boost a candidate, you should get sure you can modify it. Candidates taken from an event collections are immutable, so you need to clone them before boosting.

If you want to boost a candidate to another candidate center of mass, you can do the following:

```
Candidate * c1clone = c1->clone();
Booster boost(c2->boostToCM());
booster.set(*c1clone);
```

Once booster, if `cand1` is a `ComposteCandidate`, all its daughters are stored internally to it will also be boosted.

If you want to boost a `ComposteCandidate` and its daughters to its center of mass, you can use the following example:

```
// create booster object
CenterOfMassBooster boost(h);
// clone object and update its kinematics
Candidate * higgs = h.clone();
boost.set(*higgs);
// get boosted Z as Higgs daughters
const Candidate * Z1 = higgs->daughter(0);
const Candidate * Z2 = higgs->daughter(1);
```

In the above example, the Z daughters (leptons) will also be boosted.

Booster utilities are defined in `CMS.PhysicsTools/CandUtils`.

Common Vertex Fitter

Common vertex fitter is a "setup" operator using the Vertex Fitter tools for track collections. It requires the magnetic field map to be passes to the algorithm, which can be obtained from the EventSetup:

```
ESHandle<CMS.MagneticField> B;
es.get<IdealMagneticFieldRecord>().get(B);

CandCommonVertexFitter<KalmanVertexFitter> fitter;
fitter.set(B.product());

const Candidate & zCand = ...; // get a Z candidate
VertexCompositeCandidate fittedZ(zCand);
fitter.set(fittedZ);
```

More details on:

- Constrained vertex fit with candidates

Candidates and Monte Carlo Truth

Candidate used to represent Monte Carlo truth from generator output can be matched to RECO objects using a set of common tools described in the document below:

- MC Truth matching tools

Review status

Reviewer/Editor and Date	Comments
JennyWilliams - 11 July 2006	migrated page from ParticleCandidates name, changed link from May06 tutorials to the WorkBook version MakeAnAnalysis
AnneHeavey - 13 December 2007	Most recent editing by author, Luca Lista

Responsible: LucaLista
Last reviewed by: PetarMaksimovic - 28 Feb 2008

This topic: CMSPublic > WorkBookParticleCandidateUtils
Topic revision: r31 - 2010-04-27 - RogerWolf