Table of Contents

Combinatorial Analysis using Candidates

Complete:

Purpose

Perform combinatorial analysis using particle Candidates. The tools avoid double-counting and don't combine particles sharing the same component in one or more subdetectors.

Particles are composed according to a decay tree, to create `CompositeCandidate` objects. Daughter particles kinematics is copied into the composite object, and links to the orignlal "master" particles can also be stored using `ShallowCloneCandidate`.

CandCombiner Framework Modules

Different modules are provided, with possibility to read and save different types of collections. The supported combiner modules are specialization of a single module template, `reco::modules::CandCombiner`, defined in the package:

- `CommonTools/CandAlgos`

The type of decay is specified passing a string like in the following examples, where the character ' `@` ' is used to specify the charge values after the collection name (zero charge is assumed if missing):

- `decay = cms.string("muons@+ muons@-")` for Z $\mu^+\mu^-$
- `decay = cms.string("pions@+ pions@- pions")` for + - 0
- `decay = cms.string("jPsi phi")` for B_s J/

Charge conjugate decays are always implied.

Examples of specializations for this modules are:

CandViewShallowCloneCombiner

The module CandViewShallowCloneCombiner combines particle candidates to form composite objects. Daughter particles are shallow clones of the input particles. The cut is specified as a string.

- **Input collections** can be any type containing candidates (`edm::View<reco::Candidate>`)
- **Output collection** is of type `std::vector<reco::CompositeCandidate>`

Examples of configuration is:

```
process.ZCandidates = cms.EDProducer("CandViewShallowCloneCombiner",
  decay = cms.string("muons@+ muons@-"),
  cut = cms.string("86.0 < mass < 96.0")
)
```

More information on configurable string cuts can be found in:

- Physics Cut Parser.

CandViewCombiner

The module CandViewCombiner combines particle candidates to form composite objects. Daughter particles are clones of the input particles. The cut is specified as a string.

- **Input collections** can be any type containing candidates (`edm::View<reco::Candidate>`)
- **Output collection** is of type `std::vector<reco::CompositeCandidate>`

Examples of configuration are:

```
process.ZCandidates = cms.EDProducer("CandViewCombiner",
  decay = cms.string("muons@+ muons@-"),
  cut = cms.string("86.0 < mass < 96.0")
)
```

More information on configurable string cuts can be found in:

- Physics Cut Parser.

DeltaPhiMinCandCombiner

The module DeltaPhiMinCandCombiner produces combined candidate of pairs of particles (e.g.: jets) that have an acoplanarity angle Δ larger than a given configurable value `deltaPhiMin`. This module is mainly used for min-bias and underlying event reconstruction.

DeltaRMinCandCombiner

The module DeltaRMinCandCombiner produces combined candidate of pairs of particles (e.g.: jets) that have a value of ΔR larger than a given configurable value `deltaRMin`. This module is mainly used for min-bias and underlying event reconstruction.

CandCombiner Utility UPDATED

The template `CandCombiner` is defined in the following package:

- `CommonTools/CandAlgos`

Template Parameters

The utility has 5 template type arguments, some of which are optional: `CandCombiner<Selector, OutputCollection, PairSelector, Cloner, Setup>`. The argument types are:

- `Selector`: Selector type specifying which selection. The interface the one specified for the Generic Selector toolkit.
- `OutputCollection`: output collection type. Defaults are `reco::CompositeCandidateCollection` if `InputCollection` is `edm::View<reco::Candidate>` and `reco::CandidateCollection` if `InputCollection` is `reco::CandidateCollection`.
- `Cloner`: helper class to specify the cloning of daughter candidates. It is by default `combiner::helpers::NormalClone`, in which case the daughter candidates are stored in the composite candidates as clones of the original ones. It is also possible to specify `Cloner` as `combiner::helper::ShallowClone`, in which case daughters particles are shallow-clones of the original ones, which means that contain a new kinematics, with a reference to the original ("master") clone.

- `Setup`: specifies the way to set up the composite candidate kinematics. By default it is `AddFourMomenta`, which sums the momenta four-vectors. Other setup utilities could be specified, if needed (e.g.: a common vertex fitter).

CandCombiner Initialization via Constructor Parameters

Some optional information can be supplied to the `CandCombiner` constructor to specify how it should operate to perform the combinatorial analysis. The following are the possible information to pass:

- **The charges of the daughter particles**. This implies that the decay is accepted only if the daughters have the charges corresponding to the specified valies, or their charge conjugates. For instance, specifying charges `+1, -1`, only decays into oppositely charged particles are searched for; specifying charged `+1, -1, 0` could be the proper way to search for a $^{+}$ $^{-}$ ^{0}. If not charges are specified, the combinatorial analysis will not check for particle charged. The number of charged particles to be reconstructed must match the number of charge values passed.
- **The value of the selector of type `S`**. If not passes, the default constructor of `S` is used.
- **The value of the setup of type `Setup`**. If not passed, the default constructor of `Setup` is used

Different constructors are provided with the possibility to pass or not the arguments of the types specified above. All the constructor are visible in the source file:

- `CandCombiner.h`.

Performing the Combinatorial Analysis

After creating a `CandCombiner`, it can perform combinatorial analysis on any candidate collection. The candidate collections should be passed via references of type `reco::CandidateRefProd` (a.k.a.: `edm::RefProd<reco::CandidateCollection>`). If the same reference is passed more than once, the module automatically avoid double-counting. The following methods are provided:

1. `combine(const reco::CandidateBaseRefProd &) const`: performs a search for a two-body decay of candidates all belonging to the same collection.
2. `combine(const reco::CandidateBaseRefProd &, ...) const`, with two to four references to collections arguments: performs a search for a two-body, three-body or four-body decay of candidates belonging to the specified collections.
3. `combine(const std::vector<reco::CandidateRefProd> &) const`: performs a search for a *N* -body decay, *N* being the size of the passed `std::vector<reco::CandidateRefProd>`.

In all cases, if a reference to the same colleciton is passed more than once, the particle decays are never double-counted. I.e.: if the decays **X A+B** is reconstructed, the decay **X B+A** is **not** added to the return collection.

The collection is returned as an auto-pointer of type:

- `std::auto_ptr<reco::CompositeCandidateCollection>`

Example of Usage of CandCombiner

To reconstruct decays **Z $\mu^+\mu^-$**, the following code can be used:

```
edm::Handle<reco::CandidateCollection> muons;
event.getByLabel( "allMuons", muons );
reco::CandidateRefProd muonsRef( muons );
MassRangeSelector<reco::Candidate> massRange( 70, 110 );
CandCombiner<MassRangeSelector<reco::Candidate> > combiner( +1, -1 );
```

```
  std::auto_ptr<reco::CompositeCandidateCollection> zCands = combiner.combine( muonsRef );
  std::cout << "Reconstructed " << zCands->size() << " Z candidates" << endl;
```

Using `CompositeCandidate` Roles and Names

There is the capability of using `CompositeCandidateCombiners` with additional parameters which correspond to the name of the candidate (for instance, "ZtoMuMu") and the roles of the daughters (for instance, "muon1" and "muon2"). The syntax is

```
process.zToMuMu = cms.EDProducer("CandViewShallowCloneCombiner",
    decay = cms.string('muons@+ muons@-'),
    cut = cms.string('0.0 < mass < 20000.0'),
    name = cms.string('zToMuMu'),
    roles = cms.vstring('muon1', 'muon2')
)
```

Notice the additional two parameters `name` and `roles` are specified, and will be added to the candidates.

This allows the user to navigate the hierarchy of a `CompositeCandidate` event hypothesis like

```
Candidate * muon1 = zToMuMuCand->daughter("muon1");
```

As of `2.1.0`, there is also support for "hierarchical" string access such as

```
Candidate * lepton = topCand->daughter("leptonicW")->daughter("lepton");
```

These additions work with any candidate combiners.

Review status

Reviewer/Editor and Date (copy from screen)	Comments
LucaLista - 4 Dec 2007	page author

Responsible: LucaLista
Last reviewed by: Reviewer

This topic: CMSPublic > SWGuideCandCombiner
Topic revision: r17 - 2015-11-11 - ThiagoTomei