Table of Contents

9.3.2 AssociationMap Container

Complete:
Detailed Review status

Contents

Introduction

In many cases it is convenient to associate quantities to objects existing in a collection. This is true for different reasons, some of which could be:

- to add extra information to an object without modifying its structure
- to reprocess only the associated quantities without needing to reprocess the main object
- to make it possible to drop extra information when writing to disk in order to save disk space

We provide a generic association map implementation that uses EDM persistent references (`edm::Ref<..>`) in the template:

AssociationMap maps object already existing in a collection to other objects that could either be stored in a different collection (association *by reference*) or stored inside the map itself (association *by value*).

- `AssociationMap<T>`

where `T` is one of several possible helper classes described below.

Different Types of AssociationMap

Different types of AssociationMap are supported and can be used for specifying different types of the template argument `T` in `AssociationMap<T>`.

The supported types for the parameter `T` are:

- `edm::OneToValue <CKey, Val, index>`: associates an object of type `Val` to an object in a collection of type `CKey`. The associated object is contained *by value* in the map. The reference to the object in the collection of type `CKey` is stored as an index of type `index`, which, by default, is the type `unsigned int`. Shorter indices can be used for collections with a small number of objects in order to save disk space.
- `edm::OneToOne <CKey, CVal, index>`: associates an object in a collection of type `CVal` to an object in a collection of type `CKey`. The association is stored *by reference* in the map. As before, the reference to the object in the collection of type `CKey` is stored as an index of type `index`.

- `edm::OneToMany <CKey, CVal, index>`: associates many objects in a collection of type `CVal` to an object in a collection of type `CKey`.
- `edm::OneToManyWithQuality <CKey, CVal, Q, index>`: associates many objects in a collection of type `CVal` to an object in a collection of type `CKey`. The association is stored *by reference* in the map in conjunction with an object of type `Q` that is intended to measure the quality of the match. The class `Q` should support the operator " < ".

Data Storage in an AssociationMap

With the exception of data stored by value in `AssociationMap<OneToValue>` type, references to objects are stored as indices that are of type `unsigned int` by default. But these indices can be of different types if they refer to collections with a small number of objects, and this may help save disk space. For instance, if the collection has less than 65536 objects, you can use an `unsigned short`, if the collection has less than 256 objects, you can use `unsigned char`.

A reference to the collection as a whole (`edm::RefProd<...>`) contains the product identifier of the collection (again an unsigned integer index). This is stored only once in the `AssociationMap`.

So, the total size of a map containing **N** associations is determined by:

- **one** product identifier for the main collection of type `CKey`
- **one** product identifier for the associated collection of type `CVal` (not stored for `OneToValue` map)
- **N** indices for the associated objects in the collection of type `CKey`
- either:
  - **N** objects of type `Val`, for `OneToValue` map
  - **N** indices for the associated objects from the collection of type `CVal` for `OneToOne` map
  - **N** vectors of indices for the type `OneToMany` map
  - **N** vectors of indices plus quality `Q` pair for the type `OneToManyWithQuality` map

AssociationMap Interface

The `AssociationMap` interface is designed in a way similar to `std::map`, from the Standard C++ Library.

Whenever the associated object is accessed by reference, a persistent `edm::Ref<..>` is used in place of a C++ reference.

An example of how to fill and retrieve objects from an `AssociationMap` is the following, that associates multiple references to tracks to jets:

```
typedef
  edm::AssociationMap<
    edm::OneToMany<CaloJetCollection, TrackCollection>
  > JetTracksMap;

JetTracksMap map;

CaloJetRef rj = ...;
TrackRef rt = ...;
map->insert( rj, rt );
TrackRefVector tracks = map[ rj ];

JetTracksMap::const_iterator i = map.find( rj );
assert( i != map.end() );
const CaloJetRef & jet1 = i->key;
```

```
const TrackRefVector & tracks1 = i->val;
```

AssociationMap Interactive Access

AssociationMap provides two method called `keys()` and `values()` that return purely transient vectors filled with object pointers (or values for OneToValue maps). For instance, if you have a branch called "assoc" containing an association of type:

```
AssociationMap<OneToOne<TrackCollection, SuperClusterCollection> >
```

you could plot the track momentum vs super-cluster energy with:

```
Events.Draw("assoc.keys().pt():assoc.values().energy()");
```

WARNING: the above example in recent releases gives problems, probably because a ROOT bug. An error message like the following can be produced:

```
root [9] Events.Draw("electronMatch.keys()")
Error: class,struct,union or type constreco not defined  (tmpfile):1:
Error: class,struct,union or type constreco not defined  _vector.h:49:
Error: Illegal pointer operation (tovalue) (tmpfile):1:
*** Interpreter error recovered ***
```

This has been fixed in ROOT, will be released in december 2007 release.

In order to use the above interactive ROOT command, you need to create, in addition to the dictionary of the AssociationMap, also a dictionary for the types:

```
std::vector<const reco::Track *>
std::vector<const reco::SuperCluster *>
```

The following transient vector types are returned by the methods `keys()` and `values()` respectively, and require a dictionary if you want to allow interactive access:

- for `AssociationMap<OneToOne<K, V> >`:

  ```
  std::vector<const K::value_type *>
  std::vector<const K::value_type *>
  ```

- for `AssociationMap<OneToValue<K, V> >`:

  ```
  std::vector<const K::value_type *>
  std::vector<V>
  ```

- for `AssociationMap<OneToMany<K, V> >`:

  ```
  std::vector<const K::value_type *>
  std::vector<std::vector<const V::value_type *> >
  ```

- for `AssociationMap<OneToManyWithQuality<K, V, Q> >`:

  ```
  std::vector<const K::value_type>
  std::vector<std::vector<std::pair<const V::value_type *, Q> > >
  ```

Generating AssociationMap Dictionaries

If you wish to store an `AssociationMap` in the event, it is mandatory to define a dictionary of the map type you are using.

In order to create a dictionary of an `AssociationMap` type, the following guidelines should be followed:

- references to products (collections) are stored using the template type `edm::helpers::KeyVal<CKey, CVal>` for all maps except for `OneToValue`, which needs one one reference, and uses the template `edm::helpers::Key&CKey>`. Those template specializations should be added to the dictionary
- the internally stored map type should be added added to the dictionary if not already defined in `DataFormats/Common` library. In particular:
  - `AssociationMap<OneToValue<CKey, Val, index> >` requires `std::map<index, Val>` that is already defined in `DataFormats/Common` for some trivial cases of the type `Val`
  - `AssociationMap<OneToOne<CKey, CVal, index> >` requires `std::map<index, index>` that in most of the cases is already defined in `DataFormats/Common` library
  - `AssociationMap<OneToMany<CKey, CVal, index> >` requires `std::map<index, std::vector<index> >` that in most of the cases is already defined in `DataFormats/Common` library
  - `AssociationMap<OneToManyWithQuality<CKey, CVal, Q, index> >` requires `std::map<index, std::vector<std::pair<index, Q> > >`
- the type `edm::AssociationMap<...>` should be added declaring the field `transientMap_` as transient data member
- the wrapper `edm::Wrapper<edm::AssociationMap<...> >` should be added, as for any EDM type

The following dictionary is also required for `OneToValue` map:

- `edm::helpers::KeyVal<edm::Ref<CKey>, Val>`

The table below summarises the internally stored map type for the different association map types.

map type	required internal map	availability of dictionary in DataFormats/Common
`OneToValue<CKey, Val, index>`	`std::map<index, Val>`	defined for `Val` identical to `index` and of type `undigned long`, `unsigned int`, `unsigned short`
`OneToOne<CKey, CVal, index>`	`std::map<index, index>`	defined for `index` of type `undigned long`, `unsigned int`, `unsigned short`
`OneToMany<CKey, CVal, index>`	=std::map<index, std::vector<index> >	defined for `index` of type `undigned long`, `unsigned int`, `unsigned short`
`OneToManyWithQuality<CKey, CVal, Q, index>`	`std::map<index, std::vector< std::pair< index, Q > > >`	not available

An example of dictionary generation is the following. It associates many tracks to a jet, and is inspired by `DataFormats/BTauReco`:

```
<lcgdict>
  <class name="edm::helpers::KeyVal<edm::RefProd<std::vector<reco::CaloJet> >,
                       edm::RefProd<std::vector<reco::Track> > >" />
  <class name="edm::AssociationMap<edm::OneToMany<std::vector<reco::CaloJet>,
                       std::vector<reco::Track>, unsigned int > >">
    <field name="transientMap_" transient="true" />
  </class>
  <!-- the dictionary for std::map<unsigned int, std::vector<unsigned int> > is not needed
         because it is defined in DataFormats/Common library -->
</lcgdict>
```

Review status

Reviewer/Editor and Date (copy from screen)	Comments
AnneHeavey - 12 Oct 2006	moved page to workbook; minor edits and major questions!
LucaLista - 11 Oct 2006	created page

Responsible: LucaLista
Last reviewed by: PetarMaksimovic 28 Feb 2008

This topic: CMSPublic > WorkBookAssociationMap
Topic revision: r21 - 2010-01-18 - KatiLassilaPerini