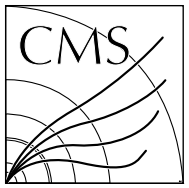
Available on CMS information server

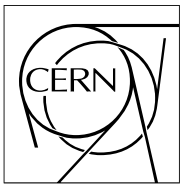
CMS NOTE-2008/015



The Compact Muon Solenoid Experiment

CMS Note

Mailing address: CMS CERN, CH-1211 GENEVA 23, Switzerland



28 April 2008

Physics Analysis Tools for the CMS experiment at LHC

Francesco Fabozzi, Christopher D. Jones, Benedikt Hegner, Luca Lista - On behalf of the CMS Offline Project

Abstract

The CMS experiment is expected to start data taking during 2008, and large data samples, of the Petabytes scale, will be produced each year. The CMS Physics Tools package provides the CMS physicist with a powerful and flexible software layer for analysis of these huge datasets that is well integrated in the CMS experiment software. A core part of this package is the Candidate Model providing a coherent interface to different types of data. Standard tasks such as combinatorial analyses, generic cuts, MC truth matching and constrained fitting are supported. Advanced template techniques enable the user to add missing features easily. We explain the underlying model, certain details of the implementation and present some use cases showing how the tools are currently used in generator and full simulation studies as preparation for analysis of real data.

1. Introduction

CMS has redesigned its software framework and Event Data Model (EDM) in 2005. This change necessitated the porting of all the existing software in order to be integrated with the new framework and EDM. It also provided an opportunity to redesign many of the existing software components, in particular the data formats and Physics analysis algorithms. All data format types and the event content have been redesigned with new EDM, in order to allow a flexible event output configuration for analysis needs. A new modular layer of software has been designed and implemented that provides a toolkit for many common analysis tasks. These utilities cover both large scale analysis processing and the final stage where the analysis is more refined. Intermediate processing output can be stored in the event in addition to the standard data formats at different processing steps, allowing the analysis to be made modular in a way that fits well with the CMS distributed computing model. A part of the analysis work-flow can run as central production, and later stages can run in local sites, providing flexibility to fulfill a rich variety of analysis processing paths.

2. CMS data tiers

2.1. CMS Event Data Model

The CMS Event Data Model[1, 2] is a uniform format and technology to manage and store all event data. An event is a container of many products, each of them can be implemented in almost any C++ type. Many event products are collections of objects (such as tracks, clusters, particles etc.). In the CMS EDM the persistent and transient representation of any data are identical. This allows uniform object access in both batch and interactive mode[3]. ROOT I/O[4] is chosen as the underlying technology for implementing the event store and Reflex dictionaries[6] are provided for every stored object type.

2.2. Event data tiers

CMS defines different standard data tiers corresponding to different levels of detail required by various applications. These range from alignment, calibration and detector studies, to Physics analysis:

- FEVT: contains the full event content, including many outputs produced by intermediate processing steps.
- RECO: contains the output of the reconstruction including enough information to be able to apply new calibrations and alignments and reprocess many of the components.
- AOD (Analysis Object Data): this is a subset of the reconstructed data (RECO) chosen to satisfy the needs of a large fraction of physics analysis studies and which should be contained in a size of roughly 100 kilo-bytes per event[5]
- TAG: this contains very basic tag information. This data tier is described in the CMS Computing TDR[5], but to date it has not been implemented.

Based on these common data formats, new data formats can be defined by adding or dropping products. This is done by simply changing the job configuration without additional need for higher level programming. Analysis provides a typical use case for requiring modified event content since there is a frequent need to define custom data types that are added to the event. This avoids the need to re-compute frequently used quantities, thus saving CPU time, and allows these quantities to be used in an interactive analysis session.

The format of the AOD has been defined and is subject to changes release by release in order to satisfy requirements for both functionality and disk storage. It is likely that AOD content will evolve with time according to the evolving analysis needs, even after the experiment starts to take data.

2.3. Modular event products

In some cases reconstructed object collections are split into multiple ROOT branches, in order to have the flexibility to store the desired level of detail in each of the data tiers,. For instance track collections are split into three separate branches:

- track collections containing track parameters with covariance matrix and fit quality (chi-squared, number of degrees of freedom),
- track collections containing additional information, such as the track extrapolation at the outermost tracker layer, and references to associated hits,
- track hit collections containing only the hits associated to reconstructed tracks.

This split provides a sufficient granularity to store different levels of details in different data tiers. Given the disk space budged, we store only the first of the three branches in AOD and RECO, and the remaining two branches only in RECO. Track refitting can therefore be performed on RECO, but not on AOD, since hits are only available in the RECO data tier.

3. Particle Candidatess

Many CMS analysis applications require objects that represent reconstructed particles. The use of common Particle Candidates for analysis has been introduced in CMS with the migration to the new framework and EDM, and was also used successfully in BaBar[9]. A common Particle candidate base class, **`Candidate`**, is defined for all high level physics objects, such as muons, electrons, photons, jets, missing transverse energy and so on. The common base class stores kinematic information (vertex position, momentum four-vector), electric charge and a particle identifier. Particle candidates may contain persistent references to AOD components, such as tracks, calorimeter clusters, etc. They are also instrumented with an interface for mother-daughter navigation, and specialized subclasses are provided to represent composite particles reconstructed from multi-body decays e.g. $Z \rightarrow \mu\mu$, $H \rightarrow ZZ \rightarrow \mu\mu ee$, $Bs \rightarrow J/\psi\phi \rightarrow \mu\mu KK$,. When cloning particle candidates, it is possible to store internally a reference to a "master" clone ("shallow" cloning). Navigating to the master clone gives the possibility to retrieve information that is associated (for instance via a reference map) to the master particle, but which is stored externally, such as isolation and jet flavour tagging,. All properties of the "shallow" clones are taken from the "master" clone, except for the kinematics. The kinematics can be updated, for instance in order to apply energy correction factors or constrained fits, keeping the original kinematics unchanged in the master clone. One of the first applications where the use of a common definition for particle candidates showed its advantages was jet clustering. Taking particle candidates as a generic input, the same jet clustering code can run on many different constituent inputs, provided that all input constituent types inherit from the Candidate base class. Extending the existing jet clustering algorithms to the Particle Flow algorithms was straightforward, since objects reconstructed by Particle Flow algorithms inherit from the Candidate base class. Particle candidates are also used as compact format for generator particles in AOD's. In this case, daughter references are stored, instead of daughter clones, as for composite candidates used for multi-body decays.

Recently a new set of candidate specializations is being developed to allow extended functionality, such as storing 4-momentum and/or vertex covariance matrices for constrained fits. No change to the core code is needed in order to accommodate new specializations.

4. Common Analysis Modules

4.1. Framework modules and event products

Reconstruction and analysis code are organized as independent modules that are driven by the CMS framework. A job configuration script defines the modules to be loaded (as plugins), configures them with the provided parameter sets, and steers the module execution sequences according to the specified "paths". Each module can retrieve data from the event and can, if needed, add new products to the event. Once a product is added to the event it can't be

modified by subsequent modules. Modules can also act as event filters, stopping the processing path if a condition is not fulfilled. These filter modules are also used to implement High Level Trigger decision paths.

Modules can retrieve event products in a type-safe way specifying the collection type. A code fragment used to retrieve a collection from an event using a label is as follows:

```
Handle<MuonCollection> muons,
event.getByLabel("muons", muons);
```

The collection is identified by its type and a *tag* ("**muons**" in this case), which is typically specified as part of the module configuration. Another way to retrieve event products without specifying the collection type can be done by specifying the base class of contained (or referred to) objects. This mechanism returns a "view", which is a container having an interface very similar to **std::vector** and which allows to access pointers or persistent references to objects stored in a collection:

```
Handle<View<Candidate> > leptons;
event.getByLabel(tag, leptons);
```

Both collections of objects and collections of references are can be accessed via "views".

4.2. Generic framework modules for AOD

A uniform interface is adopted throughout all AOD and other event data format types. This means for instance that all the methods to access the transverse momentum of an object will be called **pt()**, all methods to access the pseudo-rapidity are called **eta()**, and so on. Such simple conventions allows generic programming with templates in C++ in order to write algorithms that can be applied to many object types in a simple way without the run-time penalty of calling virtual functions. A suite of generic utilities is provided with CMS software releases as part of the Physics Tools software sub-system. Many modules performing object selections and event filtering are written using a generic approach. More high-level algorithms are also being written in this way, such as the computation of isolation variables, so that can they be computed for either muons, electrons, and tracks.

The specialization of generic modules to perform specific tasks is done using template traits[8] on the basis of input and output collection types that are passed as template type parameters. Actions such as saving clones of the selected objects, or saving persistent references to the selected objects, are supported. It is also possible to save "deep" clones of the selected objects which also includes clones of the underlying constituents (such as tracks, cluster, hits, etc.). For instance, an electron selector can save the selected electrons together with clones of the constituent track and calorimetric clusters. Those options are adopted for instance for the definition of control samples selection for alignment and calibration, for which special event output formats are defined. Since the object selection definitions are decoupled from technical implementations, such as managing object references or "deep" cloning calibration and alignment experts could easily write their event sample selections without dealing with internal selector details.

Though a fully generic mechanism to select objects on the basis of any possible interesting event property is supported, the simplest, but most frequently used object selections are based on properties of the single object. It is straightforward to define a single object selection writing a function object class returning a Boolean value. Such functors can be plugged as parameters of the generic selector templates to instantiate specialized versions of selector modules. Configurable selectors that take as input a string specifying the cut are also supported. The mapping between variables and class methods is done via the Reflex dictionary[6]. A selection could be specified simply as in the following example:

```
"pt > 10 & abs(eta) < 2.4 & normalizedChi2 < 20"
```

Navigation into the object structure via method calls is also supported. So, for instance, an electron can be selected if its track has a transverse momentum above a given threshold:

```
"track.pt > 15"
```

Or, a Z→μμ can be selected if both muons are above a given transverse momentum cut:

```
"min(daughter(0).pt, daughter(1).pt) > 15"
```

The actual selection cuts can be specified as parameters that can become part of the job configuration. Examples of module instantiations for a few of the simplest cases are as follows:

```
struct PtMinSelector {
  PtMinSelector(double ptMin) : ptMin_(ptMin) { }
  template<typename T>
  bool operator()(const T& t) const { return t.pt() >= ptMin_; }
  private:
    double ptMin_;
};

typedef SingleObjectSelector<
  reco::MuonCollection,
  PtMinSelector>
  PtMinMuonSelector;

typedef SingleObjectSelector<
  reco::TrackCollection,
  StringCutObjectSelector<reco::Track> >
     // use the string based cut
  TrackSelector;
     // save a vector of clones of the
     // selected tracks

typedef SingleObjectSelector<
  reco::TrackCollection,
  StringCutObjectSelector<reco::Track>,
     // use the string based cut
  reco::TrackRefVector>
  TrackRefSelector;
     // save a vector of references
     // to selected tracks
```

The modules defined above can be configured with the following script fragments. Cuts can be either defined as floating point parameters or within a parsed string:

```
module highPtMuons = PtMinMuonSelector {
  InputTag src = allMuons
  double ptMin = 10
   # the cut type is double
}

module bestTracks = TrackSelector {
  InputTag src = allTracks
  string cut = "pt > 10 &
    normalizedChi2 < 20"
}

module bestTrackReferences = TrackRefSelector {
  InputTag src = allTracks
  string cut = "pt > 10 &
    normalizedChi2 < 20"
}
```

The most commonly used configurable selector modules are provided as part of the CMS software release, and are ready to be plugged and configured in any framework job. If new modules are needed, many of the users usually test them as "private" instantiations, then submit them for inclusion centrally such that they can be promoted for use as common tools. The reuse of common modules occurs in this way very naturally.

5. Physics analysis common tools

A growing toolkit of common utilities is being developed for particle candidates. At the moment, combinatorial finder modules are available for managing multiple input collections and automatic overlap removal. The selection of reconstructed composite particles can be done with the already mentioned generic mechanism, including the string-based selection cut. An example of the configuration of combiner modules used to reconstruct $B_s \rightarrow J/\psi\phi$ is as follows:

```
module JPsiCandidates = CandCombiner {
  string decay = "muonCandidates@+
                  muonCandidates@-"
  string cut = "2.8 < mass < 3.4"
}

module PhiCandidates = CandCombiner {
  string decay = "trackCandidates@+
                  trackCandidates@-"
  string cut = "0.9 < mass < 1.1"
}

module BsCandidates = CandCombiner {
  string decay = "JPsiCandidates
                  PhiCandidates"
  string cut = "5.3 < mass < 5.6"
}
```

Modules to compute isolation variables with a variety of algorithms are also provided. Utilities to match reconstructed candidates with generator particles, that are also stored in AOD using the common particle candidates format, are provided. This prevents the need to rewrite each time tedious code to navigate back to parent particles in decay trees. Common constrained fitter modules are also being developed. At the moment, common vertex fitters are implemented for charged particles using algorithms that extract the covariance matrix directly from the track. New specializations of particle candidates containing the covariance matrix are under development in order to cover the cases where measurement errors are not stored with AOD objects. An example would be the case of mass-constrained fits where the particle energy is measured in the calorimeters for electrons and jets. Prototypes exists for mass constrained fits using this approach. Retrieving energy resolutions from the electromagnetic and hadronic calorimeter for electrons, photons and jets is done using specialized framework services. This approach will allow these modules to be interfacedwith the Conditions Database.

6. CMS analysis workflow

The work-flow for a typical analysis study is shown in Figure 1. Data are sent from the Filter Farm to the Tier 0 centre where they are split into several primary datasets, using High Level Trigger (HLT) bit information. They are reconstructed in a first pass at Tier-0 and are then shipped in RECO and AOD formats to the Tier-1 sites. Here regular reprocessing, recalibration and new alignments can be applied.

6.1. Analysis skims

Further specific processing steps involve analysis-specific pre-selections, so-called skimming, that run at Tier-1 sites. Output skims are copied to Tier-2 centres where there is interest to store the data corresponding to specific analysis channels. The skimming process is driven by the Physics Groups and is foreseen to happen at monthly intervals. The actual granularity and complexity of the skim processing may depend on the analysis channel. Though the standard analysis data format is AOD, additional reconstruction tasks can be run and objects specific to analysis groups can be added to the standard output format during this process. In certain circumstances one may decide to drop major fractions of the standard AOD product to save disk space. Examples of specific analysis algorithms that produce user-defined data include the reconstruction of unstable particles (e.g.: $Z \rightarrow \mu^+\mu^-$ or Higgs boson candidates), the execution of customized jet clustering algorithms, and the computation of lepton isolation variables. This mechanism has being exercised during the Computing Software and Analysis Challenge in 2007 (CSA07), and will be part of the periodic central processing when the experiment finally begins to take real data. The addition of user-data was also a software feature of the BaBar experiment[7] and was used to customize the output of analysis event skims.

6.2. Analysis processing at Tier2

At Tier-2 sites the shipped skim output is available for specific analyses and further processing steps that typically involve event selection and further specialization of the event output content. Once the data samples at Tier-2 are further reduced and stored in a sufficiently compact format, they can be shipped to Tier-3 sites for final analysis. A mixture of batch and interactive analysis can be applied at this stage.

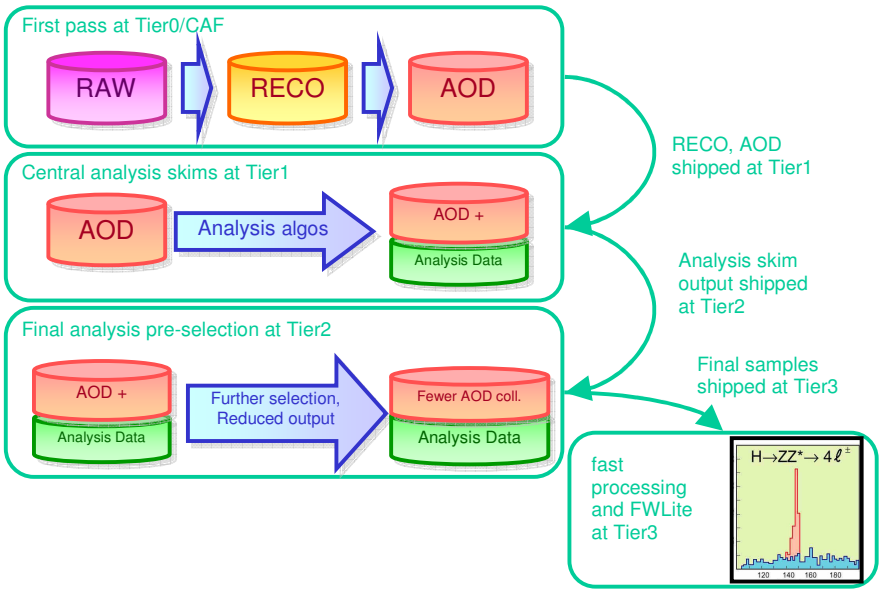


Figure. 1. A typical CMS analysis workflow.

7. Conclusions

A variety of common tools and a flexible event content have been developed in order to implement the most commonly required tasks needed for CMS analysis. The organization of data formats and tools is designed in order to be integrated with CMS analysis workflow at Tier0/Tier1/Tier2 as well as for the final stage of analysis at Tier3. A realistic exercise of analysis skims using custom data formats containing analysis collections reconstructed with common analysis modules has been prepared and will run in summer and autumn of 2007. New specializations and

extensions of the analysis toolkit described in this paper are being developed in view of the start of data taking, with the aim to cover a very large fraction of all analysis use cases.

8. References

[1] The CMS Collaboration 2007 Physics Technical Design Report, Volume II: Physics Performances, *J. Phys. G: Nucl. Part. Phys.* **34** 995-1579

[2] Jones C D et al. 2006 The New CMS Event Data Model and Framework, *Proc. CHEP 2006 (Mumbai, India, 13-17 February 2006)*

[3] Jones C D et al. 2006 Analysis Environment for CMS, *Proc. CHEP 2007 (Victoria, BC, Canada, 2-7 September 2007)*

[4] Brun R and Rademakers F, 1996 ROOT - An Object Oriented Data Analysis Framework, *Proceedings AIHENP'96 Workshop, Lausanne, Sep. 1996*, *Nucl. Inst. & Meth. in Phys. Res. A* **389** (1997) 81-86. See also http://root.cern.ch/.

[5] The CMS Collaboration 2005 CMS The Computing Project Technical Design Report, *CERN-LHCC-2005-023*

[6] Roiser S 2006 Reflex, reflection in C++, *Proc. CHEP 2006 (Mumbai, India, 13-17 February 2006)*

[7] De Nardo G and Lista L 2003 User defined data in the new analysis model of the BaBar experiment, *IEEE Nuclear Science Symposium Conference Record, Portland, OR, USA, 19-25 October 2003, Vol.* 1 pag. 165-168*; Digital Object Identifier 10.1109/NSSMIC.2003.1352022*

[8] Alexandrescu A 2001 *Modern C++ Design,* Addison Wesley Professional, ISBN 0-201-70431-5

[9] Jacobsen R J and de Monchenault G H 1998 The Beta Analysis Toolkit of the BaBar experiment, *Proc. CHEP 98 (Chicago, IL, USA, August 31 – September 4 1998)*