Physics Analysis Tools for the CMS experiment at LHC

Luca Lista[1,*], **Francesco Fabozzi**[1], **Christopher D. Jones**[2] **and Benedikt Hegner**[3]

[1] INFN, Sezione di Napoli, Complesso Universitario di Monte Sant'Angelo, via Cintia, I-80126, Napoli, Italy
[2] Cornell University, Ithaca, NY 14853, USA
[3] DESY, Deutsches Elektronen-Synchrotron, Notkestraße 85, 22607 Hamburg, Germany

Abstract. The CMS experiment will start running in 2008, and Petabytes of data will be produced every year. To make analysis of this huge amount of data possible the CMS Physics Tools package builds the highest layer of 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 like 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 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 to the core part of any software application motivated the porting of all the existing software in order to be integrated with the new framework and EDM. This phase, which is now complete, up to the ability to reproduce the algorithm performances of the old software, was also an opportunity to redesign many of the existing software components. This was in particular true for the data formats and Physics analysis algorithms. All data format types and the event content have been redesigned with new EDM, which allows a flexible event output configuration for analysis needs. A new modular layer of software providing a toolkit for many common analysis tasks has been designed and implemented. The provided 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 to modularize the analysis processing in a way to fit with distributed computing. A part of the analysis work-flow can run as central production, and later stages can run in local sites, providing flexibility to fulfil a rich variety of analysis processing paths.

[*] To whom any correspondence should be addressed. E-mail: luca.lista@na.infn.it

2. CMS data tiers

2.1. CMS Event Data Model

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 any possible C++ type. Many event products are collections of objects (like tracks, clusters, particles, …). In 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 used as underlying technology for the event store.

2.2. Event data tiers

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

- FEVT: contains (almost) the full event content, including many outputs of intermediate processing steps.
- RECO: contains a detailed reconstruction output that allows to apply new calibrations and alignments and reprocess many of the components.
- AOD (Analysis Object Data): is a subset of reconstructed data (RECO) chosen to satisfy the needs of a large fraction of Physics analysis studies.

The event content actually stored to file is fully configurable in every framework job. Storing AOD, RECO or any other custom event content as output of a job is just a matter of specializing the job configuration.

A current version of AOD has been defined and it is subject to changes release after release, under considerations of both the desired functionalities and disk size. It is likely that AOD content it will evolve with time according to the evolving analysis needs, even after the experiment will start data taking. The goal specified in CMS Technical Design Report[5] for AOD event size is 50 kilo-bytes per event.

2.3. Modular event products

In order to have the flexibility to store the desired level of detail in each of the data tier, in some cases reconstructed object collection are split into multiple ROOT branches. 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 "extra" collections*, containing additional information, like the track extrapolation at the outermost tracker layer, and references to associated hits.
- *Track hit collection*, 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 but the remaining two branches only in RECO. Track refitting, so, can be performed on RECO, but not on AOD, since hits are only available in the RECO data tier.

2.4. User-defined data tiers

Analysis groups can easily define new data types that can be added to the event for analysis studies. This is convenient to both save CPU time avoiding to re-compute quantities stored to disk, and to have the new quantities readily available for interactive analysis. The EDM allows to easily add new data types to the event, provided that Reflex dictionaries[6] are added to the new class definitions.

3. Particle candidates

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, like 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 (PDG-ID). Particle candidates may contain persistent references to AOD components, like 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, like $Z\to\mu\mu$, $H\to ZZ\to\mu\mu ee$, $Bs\to J/\psi\phi\to\mu\mu KK$, etc. When cloning particle candidates, it is possible to store internally a reference to a "master" clone ("shallow" cloning). In this way, navigating to the master clone, it is possible to retrieve information that is associated (for instance via a reference map) to the master particle, but stored externally, like isolation, tagging, etc. All properties of the "shallow" clones are taken from the "master" clone, except for the kinematics, that one can update, for instance applying 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 recently developed 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 that case, daughter references are stored, instead of daughter clones, as for composite candidates used for multi-body decays.

4. Common analysis modules

4.1. Framework modules and event products

Reconstruction and analysis code is organized as independent modules that are driven by CMS framework. A job configuration scripts defines the required 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. Those 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 to retrieve a collection from en event using a label is provided below:

```
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 product without specifying exactly the collection type can be done 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`, allowing 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 allow to use generic programming with templates in C++ 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 with a generic approach, such as the computation of isolation variables, so that can they be computed for either muons, electrons, tracks and so on.

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 like 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 (like 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, like managing objects references or “deep” cloning, experts of calibration and alignments 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 pluged 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 as simply as the following example:

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

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 given below. The code below is self-explaining:

```
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 value is specified as a 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” instantiation, then send requests to the offline group to include them centrally and promote them 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 managing multiple input collection and automatic overlap removal. The selection of reconstructed composite particle can be done with the already mentioned generic mechanism, including the string-based selection cut. An example of configuration of combiner modules can be found below, reconstructing $B_s \rightarrow J/\psi\phi$

```
    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 provided. Utilities to match reconstructed candidates with generator particles, that are also stored in AOD using the common particle candidates format, are provided, avoiding to rewrite every time tedious code to navigate back to parent particles in decay trees. Common constrained fitter modules are 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 covariance matrix are under development to cover the cases where measurement errors are not stored with AOD objects, like for mass-constrained fits where the particle energy is measured in the calorimeters, like 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 to interface, in the future, those modules with the conditions data-base.

6. CMS analysis workflow

The work-flow for a typical a analysis study is shown in Fig. 1. A first complete reconstruction processing pass is done at the Tier0 site, and the reconstructed data are then shipped at Tier1 sites in RECO (and AOD) formats. Event reprocessing can be performed at Tier1 sites applying new calibrations and alignments.

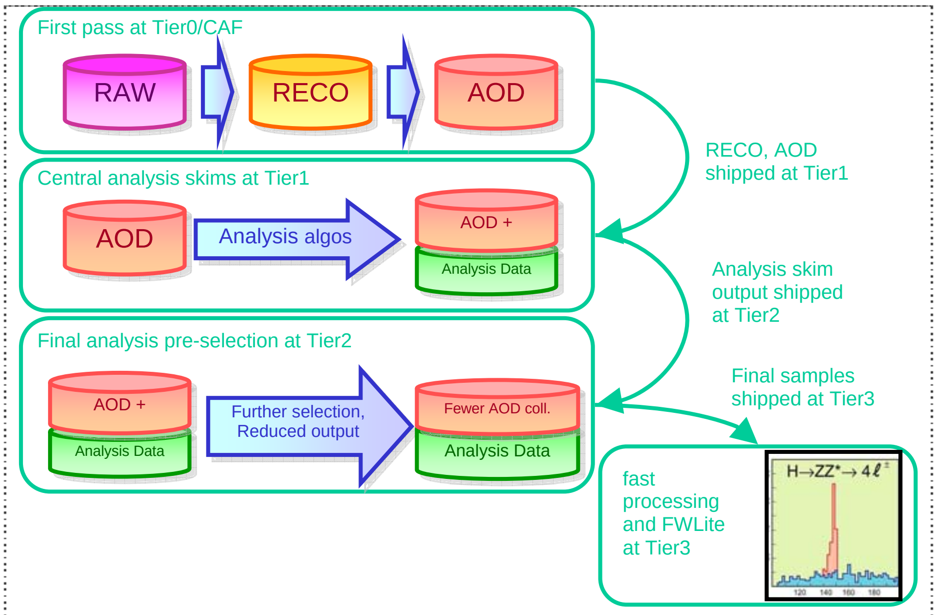
First pass at Tier0/CAF
RAW
RECO
AOD
RECO, AOD shipped at Tier1
Central analysis skims at Tier1
AOD
Analysis algos
AOD +
Analysis Data
Analysis skim output shipped at Tier2
Final analysis pre-selection at Tier2
AOD +
Analysis Data
Further selection, Reduced output
Fewer AOD coll.
Analysis Data
Final samples shipped at Tier3
fast processing and FWLite at Tier3
H→ZZ*→4ℓ±

Figure 1. A Typical CMS Analysis Workflow

Further processing steps specific for analysis are done at Tier1 and Tier2, with different time frequencies compared to full reconstruction reprocessing. The different processing steps perform subsequent event selection and data reduction steps with the desired granularity of complexity steps that may depend on the analysis channel.

6.1. Analysis skims

Analysis event pre-selections, so called “skims”, run centrally at Tier 1 sites. Skims contain both event selections and analysis algorithms specific for each analysis study, and run on the so called *primary data sets*, that are subsets of the whole CMS data sample identified by groups of High Level Trigger (HLT) bits. Skims can further select events on the basis of HLT bits and further off-line selections. The output of selected events can also be specialized for different analysis groups according to the specific needs of the different studies. Analysis groups can add their user-defined analysis collections on top of one of the standard data formats (typically AOD), or may decide to remove some collections from AOD to save disk space. The addition of user-data was also exercised in BaBar experiment[7] to customized the output of analysis event skims. Examples of specific analysis algorithms are the reconstruction of unstable particles (e.g.: $Z\rightarrow\mu^+\mu^-$ or Higgs boson candidates), but also the execution of customized jet clustering algorithms, or the computation of lepton isolation variables, etc. Presently, this mechanism is being exercised during Computing Service and Analysis Challenges, and will be part of the periodic central processing when the experiment will begin the data taking.

6.2. Analysis processing at Tier2

The output of skims done at Tier1 is shipped to Tier2 in the customized skim output formats for the different analysis studies. Further analysis processing steps, with subsequent event selection and further specialization of the event output content can be performed at Tier 2 sites. Those processing steps are managed directly by the analysis groups in coordination with the Tier 2 management. Once the data samples at Tier2 are further reduced and stored in a sufficiently compact format, they can be shipped to Tier3 sites for the final analysis processing. A mixture of batch and interactive analysis can be applied at this stage.

7. Conclusions

A flexible event content and a variety of common tools aim at implementing easily 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 is being put in production, and will run in summer and autumn 2007.

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)*

[5] 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)*