A flexible and modular data format ROOT-based implementation for HEP

Journal of Physics: Conference Series **664** (2015) 072016 doi:10.1088/1742-6596/664/7/072016

A flexible and modular data format ROOT-based implementation for HEP

Domenico D'Urso[1,2,4] **and Matteo Duranti**[2,3]

[1] ASDC, Via del Politecnico, 00133 Roma, Italy
[2] INFN Sez. Perugia, Via A. Pascoli, 06123 Perugia, Italy
[3] University of Perugia, Piazza Università 1, 06100 Perugia, Italy

E-mail: domenico.durso@pg.infn.it

Abstract. Data access and availability is a crucial issue in high energy physics (HEP) experiments, given the huge amount of data produced. We present a flexible and modular data format implementation for HEP applications. It has been designed to modularize data in order to update the minimum amount of event information in case of bug correction, software updates or data format extension, to simplify data distribution and upgrades to the regional data centers, and to reduce the amount of data to be transferred to data members really affected by reprocessing. The proposed design and implementation has been developed as mini-DST data format for the Alpha Magnetic Spectrometer (AMS [1]) experiment on the International Space Station (ISS) and is based on the CERN ROOT [2] toolkit.

1. Introduction

HEP experiments produce a huge amount of data per year, from several hundreds Tera bytes to a few Peta bytes. The access to these data is a crucial issue that has to be faced to allow the members of more and more large international collaborations, spread all over the world, to efficiently process and analyze them.

Big experiments and big collaborations, like LHC experiments, usually, are adopting a multi-tier hierarchical architecture: the data and the first level of computing (the first reconstruction of raw data) is performed on a tier 0, then these data are moved to several regional centers, tier 1, where higher level of reconstruction is performed and multiple copies of data are stored. A network of local centers, tier 2 and tier 3, allows the final users to analyze and process the data to produce physics results (see [3], [4],[5],[6] for a discussion about the computing model of mayor LHC experiments, CMS, LHCb, ATLAS and ALICE).

When a newer or better calibration is available, a bug is found in the previous reconstruction, or, in general, a new reconstruction is needed, a reprocessing is done by the tier 1 network (or tier 0, depending on the level of reprocessing needed) and then the data should be transferred again to all of the lower level tiers.

In most of the cases, only a part of the data (i.e. a single sub-detector) is affected by the new calibration or by the bug fix, while great part of the existing data does not need to be reprocessed and, moreover, to be transferred across the levels of the multi-tier structure. This constitutes a not negligible effort in terms of computing power as well as data transfer.

[4] Corresponding author

We present a flexible and modular data format that allows to face the described data issues, keeping an easy data access suitable for user analysis. The project is based on a widely used HEP toolkit, the CERN ROOT[2] (written in C++), exploiting potentialities of *TTree* class, to store large quantities of data objects, and the concept of friendship among *TTree*, to split and manage event information into a multiple file scheme. In particular, we discuss the implementation for the Alpha Magnetic Spectrometer (AMS [1]) experiment (see [7] for a discussion about AMS computing model).

2. A modular data format

The desired design for the data format should then have the following features:

- modularity, event information must be split in different files as much as possible (the minimum scheme is the splitting by sub-detector);
- distinction between low level physical quantities, that need to be recalculated only in special cases, and high level physical quantities, that for examples might use of non-standard reconstruction algorithms and/or often might need to be recalculated after a new optimization of analyses;
- easily upgradable, must be possible to upgrade and "re-build" (if a bug has been found) even a single module or, generally, a single part of event information;
- extendable, must be possible to build and add new modules to the existing ones;
- simple, it should be suitable for user analysis.

To fulfill all of the requirements, the implementation of the data format has been based on:

- C++ inheritance and polymorphism;
- ROOT TTree friendship.

2.1. C++ inheritance

The C++ inheritance allows to define a set of base classes and successively implement several extension of each of them.

A set of base classes corresponding to all of the sub-detectors of the experiment under consideration can be defined. Let's assume that one of the sub-detectors is a Tracker and define *myTrTrack* as the class to store and access the information about recorded tracks. One can imagine to store, in this class, just a few variables, as the impact point at the center of the sub-detector, the azimuthal and the zenith angle of the track, the number of hits belonging to the track and, in case of a magnetic spectrometer (like AMS), also a raw estimation of the track curvature (i.e. of the Rigidity, that is the ratio of the particle momentum over its charge). This class can be useful for simple checks, general studies and for raw analyses. On the other hand, its overall size is quite small and contained quantities are quite stable.

A second class, *myTrTrackPlus*, can be then defined to store a larger amount of Tracker related event information. This class can contain variables like the refined value of the curvature (for example depending on the detector alignment, and so subject to change), the chi-square of the fit, the average dE/dx along the track, etc... The instances of this second class, as explained below, can be streamed on a different file, w.r.t. the file where the *myTrTrack* objects are written. Different algorithms or version of the reconstruction, for the same detector, can be stored in different "Plus" classes, so defining a *myTrTrackAPlus*, *myTrTrackBPlus* and so on.

During data processing, the user can access to the whole set of information contained in the class *myTrTrack* and the group of "Plus" classes in a transparent way, regardless of which piece of event information is contained in which class, by building on-the-fly a *myTrTrackFull* instance, inheriting from all of the considered classes and simply containing default copy constructors (see section 3.4).

2.2. Polymorphism
The C++ polymorphism allows to assign different meaning or usage to the same variable, function or object. In the present case, it allows to have a very general scheme of data access. Let's consider the class containing the root of the hierarchical data structure, let's call it *myEvent*. It must have a "link" to all of the sub-classes (i.e. to all the sub-detectors), like *myTrTrack*, to permit users to access all of the stored information. Linking the sub-classes by means of pointers (for example storing the objects using vectors, or whatever ROOT collections, of pointers), one has the flexibility to route the "link" to different objects, thanks to the C++ polymorphism. The pointer can be to a generic *myTrTrackBase*, or even to a ROOT TObject, and then actually point to *myTrTrack*, *myTrTrackPlus* or *myTrTrackFull*.

2.3. TTree friendship
The concept of TTree friendship has been adopted and exploited to allow the splitting of the same event structure across different files but keeping the root of the hierarchy, the "main" (a TTree containing the *myEvent* class), self-consistent and able to be linked to whatever set of the "advanced" classes.

The idea is to store the TTree with the *myEvent* class in one file and include in the same file three leafs: run, event number and time stamp. Then one can store on different files the TTree's with the instances of the classes holding the information related to the various sub-detectors. Using the TTree friendship, one can re-build the correct "network" of objects thanks to the indexing: objects with information *per event* will be "matched" by run and event number, while objects holding time-dependent information (like slow-control variables, cfr. 3.1) by the event time stamp.

3. Implementation
The proposed data format has been tested for the realization of a "mini-DST" version of the AMS experiment reconstructed data. In this case, data format is a different, lighter and simpler organization of the already reconstructed and ROOT based AMS data. The goals, anyhow, are exactly the same: modularity and splitting (for a faster re-distribution in case of reproduction), easy upgradability and extendability (they are used as "working ntuples" but, after 4 years of mission and $\sim$ 500 TB of full reconstructed data, their production and distribution is anyhow a big effort), and simplicity.

Merging together the ideas discussed in the previous section, the implementation has been quite straightforward. A "Main" ROOT TFile contains the TTree of the *myEvent* objects. In addition, as said before, the same file contains three leaves: run, event number and time stamp for indexing.

The *myEvent* class structure is quite simple, as shown in Fig.1:

- it is a TObject, to exploit all the features the ROOT developer team put in it. Anyhow no additional space is wasted for `fBits` and `fUniqueID` since `MyClass::Class()->IgnoreTObjectStreamer()` is called in the constructor;
- the class is instanced as a singleton, so the current event is available in each part of the code (for manageability, it is not mandatory and can be easily removed to allow multi-thread processing);
- a set of vectors of sub-event "basic" object pointers, corresponding to the *myTrTrack* class typology;
- a set of vectors of sub-event "advanced" object pointers, corresponding to the *myTrTrackPlus* class typology, stored in dedicated files;
- a pointer to object containing slow control data, stored in a dedicated file (see section 3.1);

- a pointer to a generic structure that can be used to access a customized user data object, stored in a dedicated file (see section 3.2).

Vectors to "advanced" object pointers and pointers to slow control data and user class have the instruction `//!` to set ROOT to not stream the pointed objects when streaming the *myEvent* class. Those objects are streamed on a different file "by hand".

"Advanced" objects contain a larger set of sub-detector variables and variables dependent on multiple detectors. They are not written on the disk when *myEvent* is saved and each vector is stored in a dedicated file. Vectors contain pointers to generic objects, corresponding to the *myTrTrackBase* typology, to exploit the C++ polymorphism (see section 2.2).

Figure 1. The structure of the *myEvent* class.

```
class myEvent: public TObject {

 protected:
  static myEvent* ptr;//!

 public:
  static myEvent* gethead();
  static void printptr();
  virtual void init();

  std::vector<myParticle*>          vmp;
  std::vector<myTrTrack*>           vmt;
  std::vector<myEcalShower*>        vme;
  std::vector<myTrdTrack*>          vmu;
  std::vector<myBeta*>              vmb;
  std::vector<myRichRing*>          vmr;

  std::vector<myTrTrackPlus*>       vmtp;//!
  std::vector<myTrdTrackPlus*>      vmup;//!
  std::vector<myEcalShowerPlus*>    vmep;//!
  std::vector<myBetaPlus*>          vmbp;//!
  std::vector<myRichRingPlus*>      vmrp;//!

  std::vector<myTrTrackBPlus*>      vmtbp;//!

  mySlowControl* sc;//!

  myUser1* us1;//!

 public:
  ClassDef(myEvent,30);
};
```

3.1. Slow control

An object containing data taking conditions has been included in the event data structure. Since each TTree has the three leaves for indexing (run, event number and time stamp), it can be indexes on the event time stamp, so there is no need to store slow control information on event base. In the AMS case, it specifies the International Space Station and the AMS position and orientation and other quantities related to the detector status. Information are evaluated and stored second by second. Indexing on the event time stamp, one can access to the slow control data information corresponding to the data taking conditions of the event under study.

3.2. myUser

A specific pointer to a generic *myUser* object has been included within the *myEvent* data structure. The user can implement his own object and store it on a separated file, that will be accessed though the TTree friendship. It contains the three leaves to be used for indexing (run, event number and time stamp), so the user can produce his own data stream with a dedicated selection and store only "interesting" events. It can also be used to substitute the user custom ntuples.

3.3. Data access

A "main" data file is produced storing the *myEvent* data object with all of the sub-event "basic" objects attached to. Event information from accessory files are then retrieved using the TTree friendship. Access to secondary files, containing "advanced" objects, are managed by a customized TChain, *myChain*, that loads on demand the needed objects. Once the path of a "main" data tree is passed to an instance of *myChain*, it searches for all the secondary files available at a pre-defined path, that can be set by the user, and it assigns each of retrieved objects to the corresponding sub-event "advanced" object pointer, contained in the *myEvent* object. The user can also set to retrieve only a subgroup or a specific accessory file. Through the *myChain*, the user can access to all of event information. Furthermore, exploiting the properties of its ROOT parent class, the TChain, the user can do a simple draw of desired variables. If the desired "advanced" object is not retrieved, the corresponding pointer will be kept at NULL. Dedicated members of *myChain* could give at any time the list of available "advanced" objects.

3.4. myFullObject

To simplify data access, a single common interface, *myFullObject*, obtained by means of a multiple inheritance, has been implemented to address the full set of information contained in one of the "basic" object and all of its "advanced" objects. Considering the case of *myTrTrack* and *myTrTrackPlus*, seen in section 3, at reading time, an instance of *myTrTrackFull* is built on-the-fly, deriving from both, *myTrTrack* and *myTrTrackPlus*. The user has not to care about, while developing its analysis code, if the data member is looking for is contained in the "basic" class or in one of its "advanced" object. It is enough to guarantee the presence of all the needed files (i.e. accessory files with the "advanced" objects) and then he can access to all of the data members of both, the "basic" and "advanced" objects, by means of a completely transparent unique interface.

The "full" object contains the list of "advanced" objects linked to the "main" file at the reading time, corresponding to the "basic" object under consideration. Taking the example of *myTrTrackFull*, it contains the list of the different *myTrTrackPlus* retrieved (*myTrTrackAPlus*, *myTrTrackBPlus* and so on). Hence, directly from the "full" object, *myTrTrackFull*, the user can realize which objects of *myTrTrackPlus* typology are available. Since the *myFullObject* inherits by both, the "base" and the "advanced" classes, one doesn't need to write and maintain the code needed to fill up the "full" object, it is automatically done by the copy constructor. In

that way, the proposed data format and the "full" mechanism are completely transparent to the final user and to the data format maintainer.

3.5. myInfo

In order to garanteee a full reproducibility of data production, a dedicated objet, *myInfo*, has been implemtented to keep track of all information regarding the DST production. In the AMS implementation, we store:

- date of DST production;
- name of "original" AMS reconstructed files processed to obtain the DST;
- version of the AMS general software used to acces AMS reconstructed files;
- version of the DST software;
- list of secondary files produced;
- additional notes to the production.

A reference to the *myInfo* object is attached to the TList containing user objects associated to the "main" tree. The user can get the access to the TList attached to the tree by the method GetUserInfo(). From the *myInfo* object, the user can derive which secondary files have been produced and if there special notes related to the production. Hence, just from the "main" tree, the user can have a detailed description of all the production. Each time a new piece of data is reprocessed, a new *myInfo* object is attached to the TList of the "main" tree.

4. Conclusions

A data format based on ROOT has been designed to be modular, upgradable and extendable, and to face all of the HEP data issues. In this contribution, the implementation of a "mini-DST" version of the AMS experiment reconstructed data has been presented. Event information are split over several files, that may contain different parts of event (i.e. different sub-detectors) or different level of abstraction (i.e. from raw information to the highest level quantities). Due to its modularity, it is suitable for a mini-DST format as well as a complete data format. Data set is easily upgradable, without a full reprocessing of data. File splitting allows a very efficient distribution and upgrade of data to the regional Centers, as well as the possibility to download and process a small fraction of event information, even on the user laptop.

Acknowledgments

We would like to thank Dr. Paolo Zuccon (MIT) for the precious discussions about the design and the possible implementation of the proposed data format. We would like also to thank Dr. Zhicheng Tang (IHEP) and Dr. Konstantin Kanishchev (University of Trento) for the help given in the implementation of some functionalities and for the large amount of test and debug work.

References

[1] Aguilar M *et al.* 2013, *Physical Review Letters* **110**, 141102.
[2] Antcheva I *et al.* 2009 *Computer Physics Communications* **180** 12, 2499 - 2512, https://root.cern.ch/guides/reference-guide.
[3] C. Grandi *et al.* 2014, Proc. of 20^{th} CHEP, *Journal of Physics: Conf. Ser.* **513**, 032039.
[4] M. Cattaneo *et al.* 2014, Proc. of 20^{th} CHEP, *Journal of Physics: Conf. Ser.* **513**, 032017.
[5] R. W. L Jones and D. Barberis, 2010, Proc. of 17^{th} CHEP, *Journal of Physics: Conf. Ser.* **219**, 072037.
[6] T. Chujo *et al.*, Proc. of 21^{st} CHEP, these proceedings.
[7] Shan B. *et al.*, Proc. of 21^{st} CHEP, these proceedings.