PRELIMINARY DRAFT

ROOT based analysis of the CMS data

Pavel Demin, Giacomo Bruno

February 13, 2005

Abstract

In this note information is given on design, structure and usage of the ExRootAnalysis package.

1 Introduction

The ExRootAnalysis package stores reconstructed events in a ROOT tree format [1]. ROOT allows to store data in several different formats. However there are some crucial differences, which makes the ROOT tree format more attractive than others:

- information is stored in TClonesArray, which enables efficient storage (compression using the ROOT gzip algorithm) and retrieval;
- same C++ classes (TRootJet, TRootElectron *etc.*) used for creating the ROOT tree and for analysing the stored data;
- different quantities stored in the tree can be linked by C++ pointers.

ROOT tree objects are created from reconstructed objects (in most of the cases physics objects *i.e.* jets, electrons *etc.*), in order to perform analysis in a ROOT environment.

They can be produced in several different places of the CMS software, like using data in Digi or DST formats. The principal application of the ExRootAnalysis package is however in connection with the DST. The primary goal is to give the possibility to include in the ROOT tree the objects that are present in the DST. These should be just those that are indispensable for the physics analyses. The package can be easily extended with new branches *i.e.* to store additional information relevant to a given physics analysis.

In Section 2 the package structure is briefly described. The contents of the ROOT tree is summarised in Section 3. In Section 4 one can learn how to create his/her own ROOT tree. Finally, Section 5 describes the way one can analyse the ROOT tree.

2 Structure of the ExRootAnalysis package

The ExRootAnalysis package can be subdivided in the following subsystems:

basic framework: few classes providing event loop, event selection and basic operations with a ROOT tree file;

blocks: modules writing information into ROOT tree branches

selectors: modules selecting events to be analysed stored in the ROOT tree file;

services: modules providing functionality common to several blocks and selectors.

The package steering classes are UserRootAnalysis and RootAnalysis. RootAnalysis is an Observer of the G3EventProxy. It basically provides the event loop, handles the ROOT tree and provides a number of services for the specific user analysis. One can tailor his/her own analysis by only modifying the constructor of the UserRootAnalysis class and the .orcarc configuration file.

The system is structured in such a way that there is a one to one correspondence between classes describing branch data and DST classes. For example for RecJet the corresponding class is TRootJet, for ElectronCandidate it is TRootElectron, *etc.* For every TRootXXX class there is a corresponding BlockXXX class. All TRootXXX classes inherit from the TObject. All BlockXXX classes inherit from BlockBase. The first action that the user has to perform is to add the blocks necessary for the analysis as it is done in the released version of UserRootAnalysis.

Two links services are provided at the moment through the class ServiceFactory. One of them can be used to create generic links among the tree branches by making use of the persistent DST objects. For example the TRootElectron class contains a link (pointer) to the corresponding TRootTrack class. The second is a similar link service, provided for the calorimeter cell-related blocks.

Finally, the user can add event selectors in order to filter the events of the POOL collections to be analysed. Two standard event selectors are provided: SelectorL1 and SelectorHLT. As the name says, they filter events according to the L1 trigger and HLT global decision, respectively. It is important to note that the order in which selectors are added determines the order in which they are executed. If one of them fails to select the event, the following ones will not be executed and the program will skip the event.

Every BlockXXX has a method called accumulate(G3EventProxy*) that loops over all entries in a RecCollection of a specified type and creates corresponding TRootXXX instances by calling newEntry() method. This method automatically puts those TRootXXX objects on the appropriate branch.

ExRootAnalysis creates all the BlockXXX instances according to configuration specified in a .orcarc file. The configuration file has a set of Boolean parameters to decide which blocks should be created. For some blocks more parameters could be specified like reconstruction algorithm, maximal number of entries per event, *etc.*

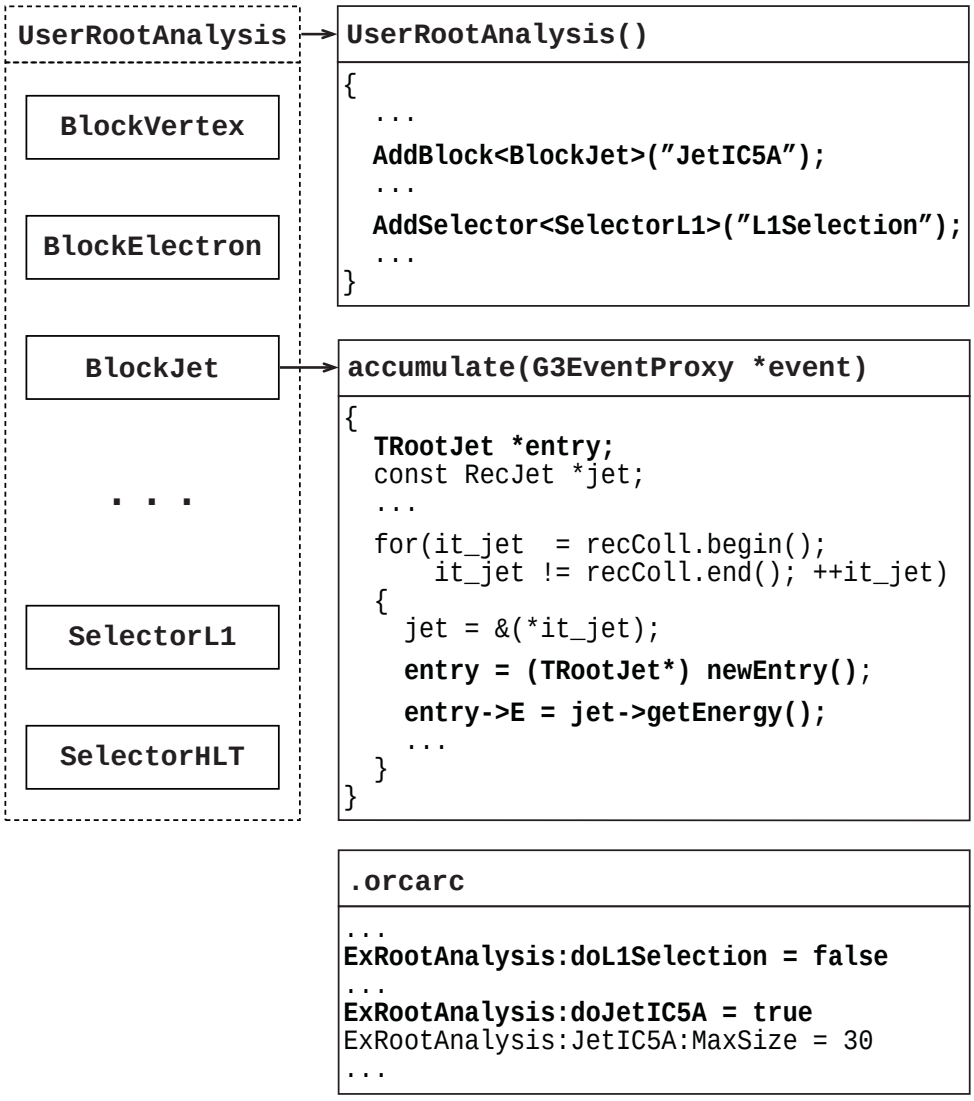

Figure 1: Structure of the `ExRootAnalysis` package

3 Contents of the ROOT tree

Table 1 shows the branches available at present together with the reconstructed objects they correspond to. The third column shows the names of the corresponding classes to be written in a ROOT tree. For uniformity, each branch is represented by a TClonesArray. If a branch contains a single entry per event (for example $\not{E}_T$), then this branch is represented by a TClonesArray with only one entry.

Objects stored in the tree are linked by means of TRef pointers or TRefArray (array of pointers). More documentation on the content of the ROOT tree is available on the web [2].

4 Creation of a ROOT tree

To create a ROOT tree, the following steps are needed:

- `cd ORCA_X_X_X/src`
- `cvs co Examples/ExRootAnalysis`
- `cd Examples/ExRootAnalysis`
- `scram b`
- `eval ‘scram runtime -csh‘`
- `ExRootAnalysis`

Note that:

- `.orcarc` file has switches to turn on and off creation of various blocks, all blocks are off by default;
- name of the output ROOT tree file is also set via `.orcarc`, the default is `test.root`.

5 Doing analysis with a ROOT tree

We provide a set of tools simplifying access to the root ROOT trees produced produced by the ExRootAnalysis package. These tools are available as a separate ORCA package Examples/ExRootAnalysisReader. The most interesting class from this package is ExRootTreeReader. This class provides simple methods sufficient to access every branch in every event stored in a ROOT tree:

- `ExRootTreeReader(TTree*)` – constructor takes ROOT tree as parameter
- `Long64_t GetEntries()` – returns total number of events stored on the tree
- `TClonesArray *UseBranch(branchName)` – returns pointer to collection of branch elements and specifies branches needed for analysis

- `Bool_t ReadEntry(entry)` – loads collections of branch elements with data from specified event

The following commands allow to install and to compile this package. Make sure that you install both packages `ExRootAnalysisReader` and `ExRootAnalysis` in the same `Examples` directory. Otherwise, `ExRootAnalysisReader` will not be able to locate some header files from `ExRootAnalysis`.

- `cd ORCA_X_X_X/src`
- `cvs co Examples/ExRootAnalysisReader`
- `cd Examples/ExRootAnalysisReader`
- `scram b`
- `eval ‘scram runtime -csh‘`

These commands produce a shared library to be loaded during a `ROOT` interactive session. The following macro illustrates a basic analysis consisting of histogram booking, event loop (histogram filling) and histogram display:

```
{
  // Load shared library
  gSystem->Load("libExRootAnalysisReader");

  // Create chain of root trees
  TChain chain("Analysis");
  chain.Add("test.root");

  // Create object of class ExRootTreeReader
  ExRootTreeReader *treeReader = new ExRootTreeReader(&chain);
  Long64_t numberOfEntries = treeReader->GetEntries();

  // Get pointers to branches used in this analysis
  const TClonesArray *branchVtx = treeReader->UseBranch("VtxPVF");
  const TClonesArray *branchJet = treeReader->UseBranch("JetIC5A");

  // Book histograms
  TH1 *histVtxZ = new TH1F("histVtxZ", "vtx position", 50, -50.0, 50.0);

  // Loop over all events
  for(Int_t entry = 0; entry < numberOfEntries; ++entry) {

    // Load selected branches with data from specified event
    treeReader->ReadEntry(entry);
```

```
    // If event contains at least 1 vertex
    if(branchVtx->GetEntries() > 0) {
      // Take first vertex
      TRootVertex *vtx = (TRootVertex*) branchVtx->At(0);
      // Plot vertex Z co-ordinate
      histVtxZ->Fill(vtx->Z);
      cout << vtx->Z << endl;
    }
  }

  // Display resulting histogram
  histVtxZ->Draw();
}
```

For more advanced example of analysis macro illustrating the most of the functionality of the `ExRootAnalysisReader` packages see

```
Examples/ExRootAnalysisReader/test/Example.C
```

6 Adding new branches

Contributors to the `ROOT` tree should use one of the existing `BlockXXX` classes as a template for their own classes. To create a new `ROOT` tree branch, the following steps are needed:

- Define new `TRootXXX`
  - objects of class `TRootXXX` must have fixed size;
  - class `TRootXXX` can contain following types of data:
    - simple types (`Int_t`, `Float_t` ...)
    - fixed size structures (`Int_t[10]`, `TLorentzVector` ...)
    - persistent pointers to other objects (`TRef`, `TRefArray`)
  - all classes `TRootXXX` are defined in `interface/BlockClasses.h`
- Include the new `TRootXXX` class into the `BlockClassesLinkDef.h` file with the following line:

  ```
  #pragma link C++ class TRootXXX+;
  ```

- Add appropriate line into `BlockClasses.cc` if objects of new class are requested to be sortable

  ```
  TCompare *TRootXXX::fgCompare = ...;
  ```

- Define new `BlockXXX` inheriting from `BlockBase` and implementing the following methods:

  - constructor `BlockXXX()`
    - specify class `TRootXXX` to be stored in branch
    - read parameters from `.orcarc`
  - `void accumulate(G3EventProxy*)` – method called once for each event
    - fill objects of class `TRootXXX` by analysing the event

- Activate new block in `UserRootAnalysis` constructor:

  `addBlock<BlockXXX>("`*`BlockName`*`");`

- Turn new block on in `.orcarc` file:

  `ExRootAnalysis:do`*`BlockName`*` = true (default is false !!!)`

References

[1] `ROOT`,
`http://root.cern.ch/`

[2] `ROOT` tree structure,
`http://cmsdoc.cern.ch/swdev/viewcvs/viewcvs.cgi/*checkout*/`
`/ORCA/Examples/ExRootAnalysis/doc/RootTree.html`

Branch	Definition	Class
Gen	generator particles from HEPEVT	TRootGenParticle
Event	general event information	TRootEvent
L1Info	L1 trigger bits	TRootL1Info
L1Elec	L1 electron candidate	TRootL1CalCandidate
L1IsoElec	L1 isolated electron candidate	TRootL1CalCandidate
HLTInfo	HLT trigger bits	TRootHLTInfo
HLTCand	HLT candidates	TRootHLTCandidate
Cal	reconstructed calorimeter hits	TRootCalCell
TrkCmb	tracker tracks (CombinatorialTrackFinder)	TRootTrack
TrkPix	pixel tracks (PixelTrackFinder)	TRootTrack
TrkEP	off-line electron tracks	TRootTrack
VtxPVF	vertices (PrincipalVertexFinder)	TRootVertex
VtxPVFPrim	vertices (PVFPrimaryVertexFinder)	TRootVertex
CalXtra	information for soft electron b-taggin studies	TRootCalXtra
CalXtraPix	information for soft electron b-taggin studies	TRootCalXtra
CalXtraEP	information for soft electron b-taggin studies	TRootCalXtra
EGBClus	e/gamma basic clusters	TRootEGBasicCluster
EGClus	e/gamma clusters	TRootEGCluster
EGSClus	e/gamma super clusters	TRootEGSuperCluster
EGEClus	e/gamma endcap clusters	TRootEGEndcapCluster
EGSCand	e/gamma off-line barrel candidates	TRootEGCandidate
EGECand	e/gamma off-line endcap candidates	TRootEGCandidate
Elec	off-line electron candidates	TRootElectron
Photon	off-line photon candidates	TRootPhoton
MuonL2	L2 muons	TRootTrack
MuonL3	L3 muons	TRootTrack
MuonSTA	stand alone muons	TRootTrack
MuonISO	isolated muons	TRootTrack
MuonGLB	global muons	TRootTrack
GenJetIC5A	iterative cone (0.5) jets from generator particles	TRootJet
JetIC5A	iterative cone (0.5) jets	TRootJet
JetIC5B	iterative cone (0.5) calibrated (JetPlusTrack) jets	TRootJet
JetIC5C	iterative cone (0.5) calibrated (GammaJet) jets	TRootJet
JetIC7A	iterative cone (0.7) jets	TRootJet
JetKT4	KT recom (4) jets	TRootJet
JetKT1	KT recom (1) jets	TRootJet
METMC	METfromParticle	TRootMissingET
METL1	L1TriggerMET	TRootMissingET
METCT	METfromCaloRecHits	TRootMissingET
METCH	METfromEcalPlusHcalTower	TRootMissingET
METIC	METfromICJet	TRootMissingET
METKT	METfromKTJet	TRootMissingET

Table 1: ROOT tree contents