The tmb_tree package

DØnote 3978

Serban PROTOPOPESCU
BNL Brookhaven
Stéphanie BAFFIONI, Elemér NAGY, Eric THOMAS
CPPM Marseille
Nadia LAHRICHI
CEA Saclay

Abstract

In this note information is given on design, structure, coding and use of the tmb_tree package.

1 Introduction

The tmb_tree package stores reconstructed events in a ROOT Tree format [1], hereafter called TMBTree. In many respect this data format is similar to that used presently as ROOT HEP-Tuples produced by Reco_Analyze[1]. However there are some crucial differences, which makes the ROOT Tree format more attractive than HEP-Tuples:
(*i*) Information is stored in TClonesArray [1], which enables efficient storage (compression using the ROOT gzip algorithm) and retrival;
(*ii*) The same C++ classes used for creating the tree have to be used in analyzing it;
(*iii*) Different quantities stored in the tree can be linked by C++ pointers;
(*iv*) One can take advantage of new features of ROOT in contrast with the ROOT HEP-Tuple, which is not being developped anymore (only maintained).

TMBTree Objects (like DØ HEP-Tuple Arrays) are created from reconstructed objects (in most of the cases Physics Objects), in order to perform analysis in a ROOT environment.

[1]The authors of tmb_tree deliberately maintain the same variable names everywhere when possible as used in the ROOT HEP-Tuple

They can be produced in several different places of the DØ software, like using DØReco output, or inside PMCS. The principal application of the tmb_tree package is however in TMB_Analyze, i.e. in connection with the Thumbnail [2]. The primary goal is to include in the tree all objects that are present in the DØ HEP-Tuple and indispensable for the analyses. However, the package can be easily extended to new branches i.e. to storage of additional objects.

In Section 2 the content of the TMBTree is summarized. The program structure is described in Section 3. In Section 4 one can learn how to create his/her own tree. Finally, Section 5 describes the way one can analyze the TMBTree. The steps necessary for TMBTree analysis will be simplified in the future.

2 Information content of the TMBTree

Table 1 displays the branches introduced at present together with the reconstructed objects they correspond to. Trigger information is limited for the moment to names of triggers that fired, but will be extended soon. The third column shows the file name where the corresponding C++ Class is defined (implemented). If a branch contains a single entry per event (for example MissingET), then this branch is represented by a simple class. If a branch contains multiple entries (for example several jets in one event), then this branch is represented by a TClonesArray. If for one type of object there are several Physics Chunks (for example jets reconstructed using several algorithms), only one branch is created and a word is added to each reconstructed object as a label indicating the algorithm used. An option to select objects by algorithm using this label during the creation of the tree will be added eventually (probably using rcp switches). Objects stored in the tree are linked by TRef pointers or TRefArray array of pointers [1]. These links are listed in Ref [2]. More documentation on the content of the TMBTree will be available in the future. For the moment the user can consult the data members of the container classes in the subdirectory *tmb_tree/tmb_tree (mc_analyze/mc_analyze)* or the file *tmb_analyze/macros/TMBTreeClasses.hpp*, as well as the tutorial[3][4].

3 Structure of the program

The program contains two framework packages:

TMBTreePkg: Makes the TMBTree, fills it with TObjects and writes out the tmb_tree.root file. It does all the work.

Object	Branch	File	cvs package
Vertex	Vrts	TMBVrts.hpp(.cpp)	
ChargedParticle	Trks	TMBTrks.hpp(.cpp)	
	IsoTrks	TMBIsoTrks.hpp(.cpp)	
EMparticle	Emcl	TMBEmcl.hpp(.cpp)	
MuonParticle	Muon	TMBMuon.hpp(.cpp)	
Tau	Taus	TMBTaus.hpp(.cpp)	tmb_tree
Jet	Jets	TMBJets.hpp(.cpp)	
MissingET	Met	TMBMet.hpp(.cpp)	
FPSRecoCluster	PS	TMBFps.hpp(.cpp)	
CPS3DCluster	CPS	TMBCps.hpp(.cpp)	
Global info	Glob	TMBGlob.hpp(.cpp)	
Triggers fired	Trig	TMBTrig.hpp(.cpp)	
MCParticle	MCpart	TMBMCpart.hpp(.cpp)	mc_analyze
MCVertex	MCvtx	TMBMCvtx.hpp(.cpp)	
LinkedPhysObj	TRefs	TMBTRefs.hpp(.cpp)	tmb_analyze

Table 1: List of branches

TMBCorePkg: Creates all the object makers. It is a package with only a constructor. The TMBCorePkg.rcp files has a set of boolean parameters to decide which makers should be created. External contributors to the root tree should make packages like TMBCorePkg and use one of the TMBxxxMaker and TMBxxx classes as a template for their own objects. Examples of external packages are in the mc_analyze cvs package: mc_analyze/mc_analyze/TMBTreeMCPkg.hpp and mc_analyze/src/TMBTreeMCPkg.cpp for MCpart and MCvtx and in the tmb_analyze cvs package:
tmb_analyze/tmb_analyze/TMBTrefsPkg.hpp and tmb_analyze/src/TMBTrefsPkg.cpp for TRefs, respectively.

The above packages are used by TMBAnalyze_x (generated by the tmb_analyze cvs module).

Some technical details:

The system is structured in such a way that there is a one to one correspondence between TObject classes and D0PhysObj classes. For example for ChargedParticle the corresponding TObject is TMBTrks, for EMparticle it is TMBEmcl, etc. For every TMBxxx there is a corresponding TMBxxxMaker. All TMBxxx inherit from TObject.[2] All TMBxxxMaker inherit

[2]Not all TMBxxx that are created correspond to D0PhysObj's, Vertex is not a D0PhysObj nor are the preshower clusters. Those derived from D0PhysObj inherit from TObject via the virtual particle class TPhysObj to provide a uniform interface.

from TMBMaker. TMBTree contains a TList of TMBMaker's. When a new TMBxxxMaker is created it automatically adds itself to the TList in TMBTree.

TMBTreePkg::processEvent calls TMBTree::Make(Event&) which loops through TList and calls TMBMaker::Make(Event&) for every entry.

Every TMBxxxMaker has a Make(Event&) method that loops through all entries in all chunks of a specified type and creates corresponding TMBxxx instances by calling Addxxx method. This method automatically puts those TMBxxx objects on the appropriate xxx branch.

The above is a succinct summary of how the system works[3]. An added feature is the use of LinkTRefChunk to keep maps between LinkIndex and its corresponding TRef. So for example, TMBTrksMaker fills a map in LinkTrefChunk as it creates TMBTrks objects. This allows TMBxxxMaker called afterwards to find the TRef's corresponding to the LinkIndex<ChargedParticle> stored in the D0PhysObj. Note that, because a new chunk (LinkTRefChunk) is created, TMBTreePkg uses the ProcessEvent instead of the AnalyzeEvent interface.

4 Creation of a TMBTree

The principal application of tmb_tree is in TMBAnalyze, i.e. creating the ROOT Tree after having unpacked Thumbnail. The executable TMBAnalyze_x is part of the official release (released on d0mino as bin/IRIX6-KCC_4_0/TMBAnalyze_x).

To create a TMBTree, the following steps are needed:

- cd your_working_directory
- cp tmb_analyze/rcp/runTMBTreeMaker.rcp . for making a TMBTree
- TMBAnalyze_x -rcp runTMBTreeMaker.rcp -log tmb.log -input_file <your_input_file> for making a TMBTree.

Note that:

- <your_input_file> can be a ThumbNail output file (extended by .tmb) or a DØReco output file. If it is a DØReco output file, the user has to remove *unptmb*

[3] This structure was inspired by ATLFAST++, the fast MC of ATLAS [5]

from the line: *string Packages="..."* in the rcp files. The minimum content of this line is:
string packages = " geo read tmb_tree tmb_core " in runTMBTreeMaker.rcp for the creation of a TMBTree.

- tmb_analyze/rcp/TMBCorePkg.rcp file has switches implemented to turn off the creation of the various objects.
- The error logger messages will appear in tmb.log. If the -log option is not used these messages will appear on the screen instead.
- The name of the TMBTree output file is set in tmb_tree/rcp/TMBTree.rcp, the default is tmb_tree.root (the string _tree.root is always added to the supplied name).
- By design, the ROOT file is not allowed to be overwritten. So, before each run, one must remove the previous TMBTree output file, if any.

The user can recreate the executable in the following way:

- place at the top of a test release
- addpkg -h mc_analyze (optional)
- addpkg -h thumbnail
- addpkg -h tmb_tree
- addpkg -h tmb_analyze
- d0setwa
- gmake mc_analyze.lib (optional)
- gmake thumbnail.lib
- gmake tmb_tree.lib
- gmake tmb_analyze.all

5 Doing analysis with the tree

In a ROOT session first one recreates [4] the TMBTreeClasses libraries by
.L TMBTreeClasses.cpp++
TMBTreeClasses.cpp(.hpp) can be found in the subdirectory *tmb_analyze/macros*. These files contain all those classes used to create the TMBTree.
Next, one executes the user analysis macro (the default TMBTree name is "TMBTree" stored on file of name "my_tmb_tree.root"):

[4] If the library exists already it is advantageous to load it by: .L TMBTreeClasses_cpp.so in order to avoid recompilation

.L TMBTree.C (or .L TMBTree.C++)
T t("my_tmb_tree.root")
t.Loop()
An example of TMBTree.C, together with TMBTree.h can be found in *tmb_analyze/macros.*
Note the following:
Although it is not compulsory, it is advantegeous to replace the maximum sizes of the TClonesArrays in TMBTree.h, kMaxXXX, by their actual values, in order to avoid waste of memory, or waste of computing time for memory reallocation. This can be done in a separate ROOT session using the commands:
TFile *f =new TFile("my_tmb_tree.root")
TMBTree.MakeClass()
TMBTree.h created in this way will contain the correct values of kMaxXXX. They have to be manually copied into the original TMBTree_bu.h file. Once it is done, TMBTree_bu.h should replace TMBTree.h [5]
Finally, when using the command .L TMBTree.C++ one has to make sure that all needed headerfiles are included (e.g. #include "TH2h") otherwise it cannot produce an *.so* file. Note also, when using .L TMBTree.C instead of .L TMBTree.C++, in some cases one has to turn off beforehand the CINT optimisation by the command: .O 0, and this will lead to a longer execution time. More details can be found in the file *tmb_analyze/macros/README.txt*

The example of users macro TMBTree.C given in *tmb_analyze/macros* shows the essential ingredients to perform analysis on the TMBTree. Here below are reproduced some examples. More examples can be found in Ref.[4]

```
TMBTrks * trks = 0; // declaration of a track pointer
TMBEmcl * emcl = 0; // declaration of an em-object pointer
  ...
nb = fChain->GetEntry(jentry);   nbytes += nb; // reading all event
  //or
nb += b_Trks->GetEntry(jentry); // reading an entry of the track branch
nb += b_Emcl->GetEntry(jentry); // reading an entry of the em-object branch
  ...
trks = ( TMBTrks* ) fTrks->At(i); // get ith track
h1->Fill( trks->px() ); // fill histogram with ith track's p_x
  ...
trks = emcl->GetChargedTrack(); // get charged track associated to em-object
```

[5]In the near future this procedure will be automatized, using a method similar to MakeClass(), which will create directly TMBTree.h with the actual maximum values of the TClonesArrays

Note that the applied methods are provided in the corresponding container class.

The following example shows how to plot TMBTree variables in a simple root session using the Branch names and the container class methods:

```
.L TMTreeClasses_cpp.so
TFile* f1=new TFile("my_tree.root");
TMBTree->Draw("Taus.eta()","Taus.pT()>10.0");
```

6 Conclusion

tmb_tree cvs package enables one to create reconstructed data in a TMBTree format. This data format presents advantages with respect to HEP-Tuple both in storage, retrival and analysis of data. Tools are given for users to create and analyze the the TMBTree and extend its present structure to new objects.

Acknowledgements

The authors are indebted to Philippe CANAL for his constant support and help.

References

[1] http://root.cern.ch/

[2] S. Protopopescu, S. Baffioni, E. Nagy, ThumbNail: a compact data format, DØ-Note 3979

[3] http://www-d0.fnal.gov/~serban/thumbnail/tmb_tutorial.ps

[4] http://cppm.in2p3.fr/~nagy/tmb_tutorial/

[5] http://root.cern.ch/root/Atlfast.html