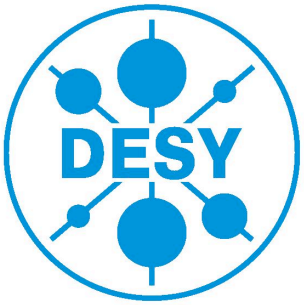


# Preparing software for simple cross-check analysis: ZeeDTTrees within the Athena framework

Alvaro Herraez Escudero

Universidad Autonoma de Madrid

DESY Summer Student Programme 2014

Supervised by Ewelina Lobodzinska

September 9, 2014

**Abstract**

This report aims to present my work during my stay at the DESY Summer Student Program 2014. The goal of the project was to prepare the necessary software in order to read the information stored in ZeeDTTrees to make it more accessible and facilitate simple cross-check analysis. In order to do that, the ATLAS framework Athena was used to access the information, which was stored in the StoreGate (the ATLAS Transient Data Store) and record it in ROOT files that could be easily manipulated. Some control plots of the $W \rightarrow e\nu$ process were obtained.

# Contents

# 1 Introduction

ATLAS is a large international collaboration of many thousand physicists from all over the world. The ATLAS experiment was designed to cover the full physics potential of the LHC and it is supposed to be in operation for two decades approximately. The ATLAS collaboration is divided into subgroups which perform different physics analyses like the measurement of the Z and W cross sections in electron and muon channels. In this project in particular, software was prepared in order to analyze the data collected in the ATLAS detector to study the $W \to e\nu$ process.

The data was stored in TTrees, whose information was recorded using ZeeD. A TTree is an object of a class that is optimized to reduce disk space and enhance access speed. A TNtuple is a TTree that is limited to only hold floating-point numbers; a TTree on the other hand can hold all kind of data, such as objects or arrays in addition to all the simple types [1]. In particular, the TTrees used in this project store events of the class ZeeDROOTEvent and the Athena framework, an enhanced version of the GAUDI framework that was originally developed by the LHCb experiment, was used to deal with them.

In this report I have first given an introduction to the Athena framework and to its Transient Data Store (StoreGate) as well as a quick overview to ZeeD, mainly applied to ZeeDROOTEvent classes. Right after that, when all the tools used in this project had been explained, I have gone deeper into the actual project and how the code was implemented to produce the control plots that can be used in the posterior analysis. It is important to remark that the project did not aim to conduct a detailed analysis of the physical processes related to $W \to e\nu$ but to do the previous work in order to prepare everything for the analysis.

# 2 The ATLAS framework Athena

Athena is the control framework used in ATLAS. It is based on the GAUDI component architecture, which was originally developed by LHCb [2].

## 2.1 Components

The major components in Athena are the following [3] [4]:

- Aplication Manager: Coordinates the activities of all other components within the application.
- Services: Provide utilities to be used by other components (e.g. by Algorithms). They are usually high-level to support the needs of the physicist. Common services are JobOptionSvc,MessageSvc, StoreGateSvc...
- Algorithms: Provide the basic per-event capability of the framework (i.e. they run once per event). Perform a well-defined, configurable operation on some input data and can produce new output data. They are written in *C++* almost exclusively.
- AlgTools: Manipulate input data to produce output data (like algorithms) but can be executed multiple times per event. Each instance of an AlgTool is owned by an Algorithm, a Service or the ToolSvc (by default).

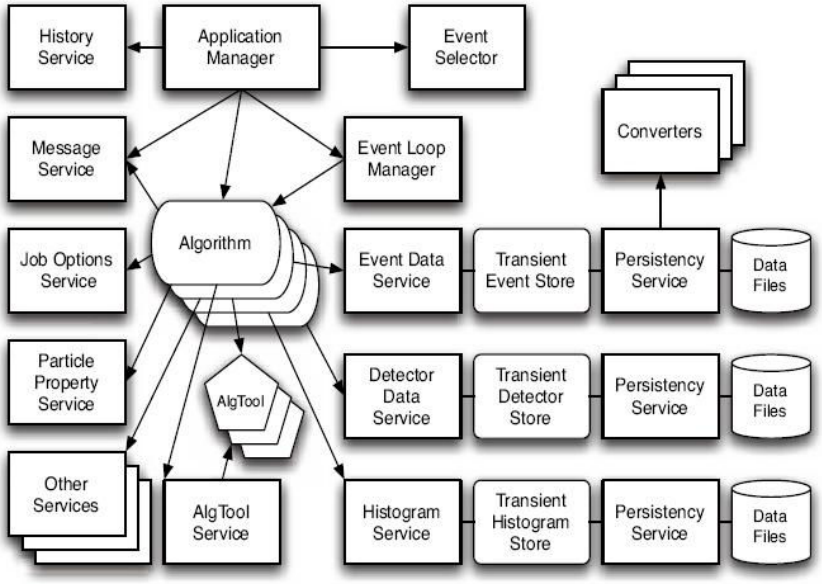


**Figure 1:** GAUDI Architecture Object Diagram

- Transient Data Stores: Data objects accessed by the algorithms are organized in various transient data stores. Depending on their characteristics and lifetimes, they are stored in different data stores.
- Selectors: They perform selections (e.g. The Event selector allows the Application Manager to select the events that the application will process)
- Converters:They convert information from one representation to another (e.g. from the transient to the persistent representation)
- Properties: All components have adjustable properties that modify the operations they perform.
- Utilities: *C++* classes that provide support for the other components

The relations of the instances of the components in terms of navigavility and usage are shown in fig. 1, where the previously explained components are shown.

## 2.2 Services

One of the most important components within the Athena framework is the Services, as they provide different utilities that can be used by all the other components. The most commonly used services are the following [3]:

- Job Options: The JobOptionSvc is a catalogue of user-modifiable properties of Algorithms, Services and AlgTools.
- Logging: The MessageSvc controls the output of messages sent by the developers using a MsgStream. The developer specifies the source of the message (i.e. its name) and verbosity level. The service can also be configured to filter out messages depending on the source or the verbosity level.

- Error handling: When an exception is thrown, it is caught and passed on to the Exception Service, which allows the users to manipulate the outcome. The service can be configured (via Job Options) to change the meaning of an exception from a failure to a success or a recoverable error.

- Performance and resource monitoring: The AuditorSvc and the ChronoStatSvc manage and report the results of a number of Auditor objects, providing statistics on the CPU and memory usage (including potential memory leaks) of Algorithms and Services.

In addition to these generic tools, Athena also provides many domain-specific tools [4]:

- HistorySvc: Records the state of all Services, Algorithms, AlgTools, and run-time parameters and environment variables at the beginning of the job, using the appropriate type of History Object.

- Histogramming and N-Tuples: Allows us to book, fill, manipulate and analyse histograms and N-Tuples from within the GAUDI framework.

- Access Time-Varying Data: Users can request a smart pointer to the appropriate conditions data, knowing that it will always be kept up to date without their intervention.

- Random Numbers: Manages random-number generation.

### 2.3 Practical aspects

In practical terms, once the code has been written and compiled, the framework produces only one executable aplication. The other components produce shared libraries that provide services for the algorithms and algorithms for data procesing (in our case, this produces all the necessary information to read the events saved in ZeeDROOTEvent -See section 4) . In addition, Athena is configured before a run starts using the job options, which are written in Python and upon execution, each of the files given to the Athena executable (athena.py) is loaded in sequence. [2]. Furthermore, three steps are always performed in a standard Athena job:

- Initialization: Services and algorithms are loaded on demand.

- Execution: Algorithms run sequentially on each event.

- Finalization: Algorithms are terminated and objects deleted.

## 3 StoreGate, the ATLAS Transient Data Store

The Transient Data Store (TDS) is the blackboard of the GAUDI architecture, and StoreGate is the ATLAS implementation of the TDS. The TDS allows a module (i.e. an algorithm, a service or a tool) to use data objects created by an upstream module or read from the disk. Among other utilities, StoreGate allows identification via data type and key string, supports base-class and derived-class retrieval via symLinks, key aliases and inter-object references and allows items to be written into ROOT files using the Athena I/O infrastructure. Once an object is posted on to TDS, it takes ownership of it and manages its lifetime according to preset policies. The TDS provides the following support for access to data objects managed by it [4]:

```
private:
    // Fields that must be saved in ROOT. NO POINTERS ALLOWED
    std::vector<ZeeDROOTElectron*>          fElecArray;
    std::vector<ZeeDROOTJet*>               fJetArray;
    std::vector<ZeeDROOTTrack*>             fTrackArray;
    std::vector<ZeeDROOTVertex*>            fVertexArray;
    std::vector<ZeeDROOTEtMiss*>            fEtMissArray;
    std::vector<ZeeDROOTGenParticle*>       fGenZBosArray;
    std::vector<ZeeDROOTMuon*>              fMuonArray;
    std::vector<ZeeDROOTMuonSpShower*>      fMuonSpShowerArray;
    std::vector<ZeeDROOTPhoton*>            fPhotonArray;
    std::vector<ZeeDROOTTau*>               fTauArray;

    Int_t       fNvtx;
    Bool_t      fNvtx_isSet;

    Int_t       fNpv;
    Bool_t      fNpv_isSet;

    Int_t       fRunNumber;
    Bool_t      fRunNumber_isSet;

    Int_t       fEventNumber;
    Bool_t      fEventNumber_isSet;

    Int_t       fLB;
    Bool_t      fLB_isSet;

    Double_t    fActualInteractionsPerCrossing;
    Bool_t      fActualInteractionsPerCrossing_isSet;

    Double_t     fAverageInteractionsPerCrossing;
    Bool_t      fAverageInteractionsPerCrossing_isSet;
```

**Figure 2:** ZeeDROOTEvent class members

- Recording a Data Object: StoreGate must be provided with a pointer to the object and with a key and this combination must be unique.
- Locking a Data Object: By default Data Objects are locked when recorded so that they cannot be modified by downstream algorithms.
- Retrieving a Data Object: Data are retrieved by type and key (using the previously mentioned unique combination). In order to do that, StoreGate sets a pointer to the requested object
- Checking if a Data object is in the store.

# 4 ZeeD

In this section, a quick overview of ZeeD will be given. However, it will focus more in the ZeeDROOTEvent and derived classes as it is the relevant part of ZeeD which was used in this project. ZeeD is an Athena-based tool, what allows it to use all Athena services. Since ZeeD is an Athena algorithm, it has an "initialize", "execute" and "finalize" part. The "Initialize" part initializes all internal variables and Athena services and loads all necessary input information, like cut thresholds, tables and histograms with efficiencies and other corrections. The "execute" part performs the event by event analysis. At the "finalization" step all histograms and trees are written to the output file. The event reconstruction with the ZeeD algorithm follows these steps (for a more detailed discussion see ref. [5]):

1. ZeeD fills all necesary information into internal variables: tracks and clusters of electrons, primary vertex info...
2. Smearing and calibration corrections are applied.
3. Calculation of electron four vectors is performed following a complex algorithm.

In our case, the events in the TTrees that wanted to be analyized were of the type ZeeDROOTEvent and can be seen in fig. 2. The most remarkable feature of the class is the fact that the information of the possible candidates is saved in different vectors of the derived classes (ZeeDROOTElectron, ZeeDROOTJet, ZeeDEtMiss...) depending on the type of candidate. This means that, for instance, the information of each of the possible electron candidates in one event is stored in a different entry of the ZeeDROOTElectron vector called fElecArray. It also contains some control information like the run number, stored in the variable fRunNumber.

# 5 Reading ZeeDTTrees

The main aim of the project was to prepare the software that is necessary to read files that included TTrees whose events were of the type ZeeDROOTEvent and this section aims to explain the basic logic and code used to do this. The steps followed by the program were:

1. Accessing the StoreGate in order to retrieve the ZeeDROOTEvent object
2. Reading the information required to perform the analysis (this information depends on the particular analysis that wants to be done) and record it in histograms
3. Writing the histograms into a ROOT file

In order to do that, the code was written in C++ and used by Athena to create the executable application. A very simple but representative part of the code can be seen in fig. 3

```
#include "StoreGate/StoreGateSvc.h"
#include "GaudiKernel/ITHistSvc.h"


StatusCode TTreeReader::initialize(){

histElec_Et = new TH1D("histElec_Et","Electron E_{t}",110,10,120);
CHECK(m_thistSvc->regHist("/ZEED/histElec_Et", histElec_Et ));

}


StatusCode TTreeReader::execute(){

if (evtStore()->retrieve(fROOTEvent,"events").isSuccess()) {
histElec_Et->Fill((fROOTEvent->fElecArray.at(a)->fCombinedFourVector.Et());
    }
}
```

**Figure 3:** Example of the basic structure of the code. The histograms are booked and registered in the initialize and filled in the execute after checking that the retrieval from the StoreGate was successful.

Furthermore, the selection of the files, the TTrees and the branches needed to be done from the Job Options file (written in python) as well as the setup of the TTree registration service and saving the histograms into ROOT files. The code used to perform this operations is shown in figure 4.
First two lines in fig. 4(a) indicate the input file(s), next four lines basically import properties and tell the event selector to take the events from the selected files. Last two lines indicate the program to read the tree called "T" and to activate all the branches (indicated with "*"). In fig. 4(b), the THistSvc is imported in the first line and the output ROOT file (called "MyZeeDAnalysis11.root")is given in the third line, with the option "RECREATE" in order to overwrite it in case it already existed.

```
ZeeDInputFiles=[/*Write path to the files here*/]
ZeeDTTreeInput = ZeeDInputFiles

from AthenaCommon.AthenaCommonFlags import jobproperties as jp
jp.AthenaCommonFlags.FilesInput = ZeeDTTreeInput

from AthenaCommon.AppMgr import ServiceMgr

ServiceMgr.EventSelector.InputCollections=jp.AthenaCommonFlags.FilesInput()

ServiceMgr.EventSelector.TupleName = "T"
ServiceMgr.EventSelector.ActiveBranches = [ '*' ]
```

(a)

```
from GaudiSvc.GaudiSvcConf import ThistSvc

ServiceMgr += THistSvc()
ServiceMgr.THistSvc.Output = ["ZEED DATAFILE='MyZeeDAnalysis11.root'
OPT='RECREATE'"]
print theApp.TopAlg
print "TEST OPTION FILE END"
```

(b)

**Figure 4:** Job Options code (written in python). **(a)** Selection of the files, TTrees and branches. **(b)** Setting up TTree registration service and saving the histograms into ROOt files

# 6 Applying the program to the $W \to e\nu$

The program was applied to the processes in which a $W$ decays into electron (or positron) and electron antineutrino (or electron neutrino): $W^- \to e^- + \overline{\nu_e}$ ($W^+ \to e^+ + \nu_e$). In order to apply the program to this process, the information about the electron four-vector and the transverse missing energy (the neutrino) were needed. These were stored in the fElecArray (of the class ZeeDROOTElectron) and fEtMiss (of the class ZeeDROOTEtMiss) so these information needed to be accessed using the program.

## 6.1 Cuts

In addition, only the events that could contain a $W$ boson wanted to be selected and different cuts were applied in order to achieve that. The cuts were implemented as *if* statements in the source code and were the following:

- $\mid VertexZ - position \mid < 150$mm. This cut was applied to analyze only the events that occurred close enough to the nominal interaction point because otherwise the background processes become too important
- At least one electron with transverse energy $E_t > 20$GeV
- If there were more than one electron per event with $E_t > 20$GeV, then only the one with the highest energy would be selected as we were looking for one $W$ boson candidate per event
- $\mid \eta_{electron} \mid < 2.47$. This is because the characteristics of the inner detector only allow this range. In addition, a region where the information was not trustable was found and rejected: $1.37 < \mid \eta_{electron} \mid < 1.52$
- Missing transverse energy $E_t^{miss} > 25$GeV
- Transverse mass of the system composed by the electron plus the missing transverse energy $M_t > 40$GeV

## 6.2 Control Plots

After reading the mentioned information and applying the cuts, the control plots shown in fig. 5 were obtained. Fig. 5(a) shows the transverse energy of the electron candidates that has passed the

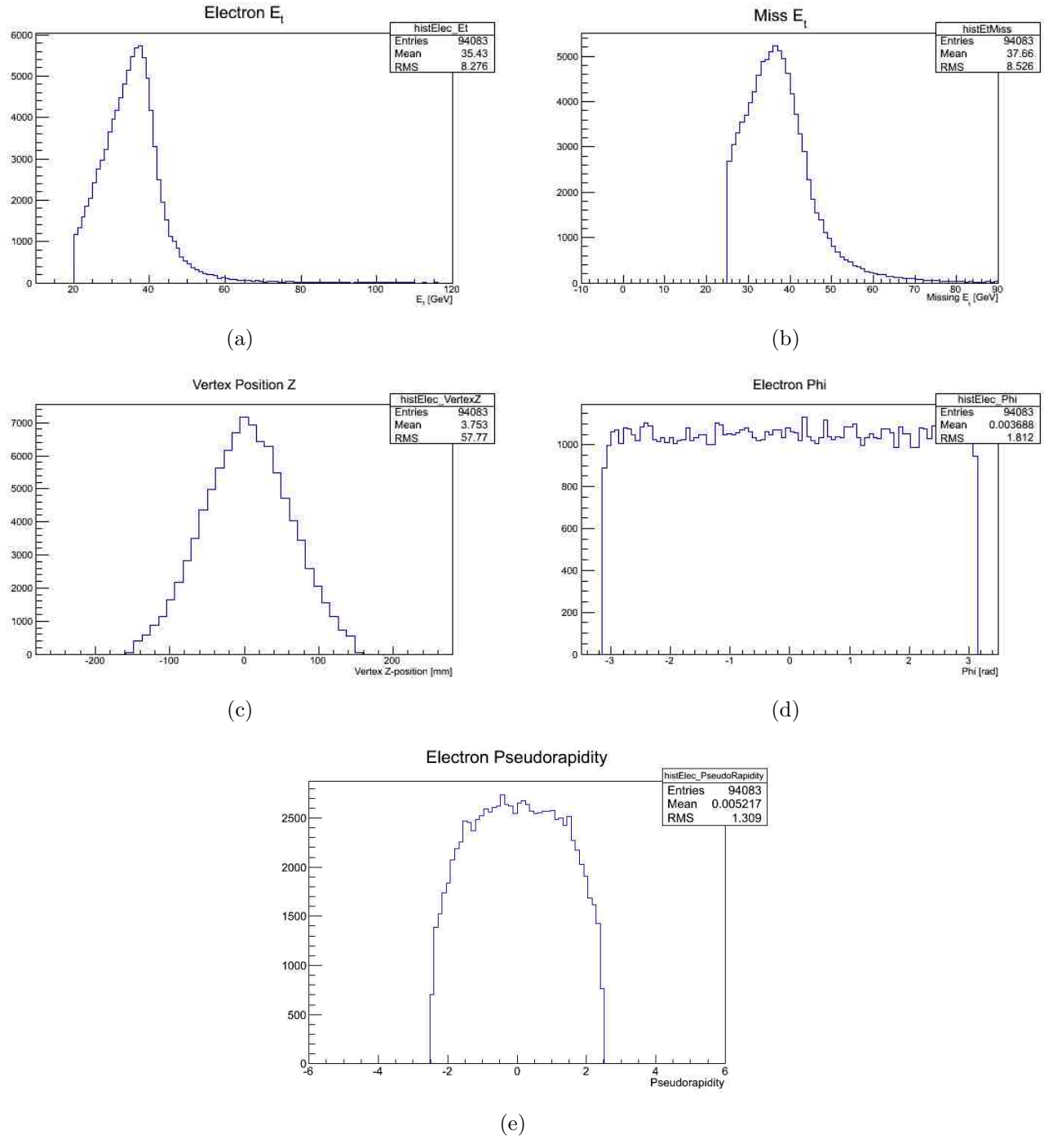


**Figure 5:** Control plots for the $W \to e\nu$ process. **(a)** Electron transverse energy. **(b)** Missing transverse energy. **(c)**Vertex Z-position. **(d)** Electron azimuthal angle in the transverse plane. **(e)** Electron pseudorapidity.

cuts and it can be seen how the cut at 20GeV was applied, as well as how the energy peaks at about 40GeV. In fig. 5(b), the missing transverse energy is plotted and both the cut at 25GeV and the peak at about 38GeV can be seen. Fig. 5(c) shows the z-position of the vertex in which the interaction

occurs and it can be seen how it has a maximum at 0 and the form of a gaussian. In fig. 5(d) the angular distribution of the electrons can be seen and its flatness indicates the isotropy of the process in the transverse plane. The last plot (fig. 5(e)) shows the pseudorapidity of the electron, where the cuts at $\pm 2.47$ can be seen.

# 7 Conclusions

After having used the program, the first thing that can be seen is that it is very easy to analyze different information (not only the one shown in the control plots in fig. 5) because what the program does is accessing the data object stored in the StoreGate so that all the information recorded in it can be used. In order to select different information to analize, the only necessary that has to be done is to declare the new histograms in the header file, book them in the initialize, and fill them in the execute. However, the main advantage of the program is the fact that it allows the performance of a simple cross-check analysis without needing to learn all the procedures and tools related to ZeeD, so it gives an fast alternative to perform this kind of analysis.

# 8 Acknowledgments

I would like to thank the whole ATLAS group at DESY as it has been a great experience to share the almost two months I have spent here with them. I would also like to thank every person who has helped me at some point, that is, James Dassoulas; Nataliia Kondrashova; George Sedov, who programmed the ZeeDROOT classes and helped us to use them; Ksenia Gasnikova, who helped me during my first weeks here and, in particular, Ewelina Lobodzinska, for spending a lot of time in order to help me and tell me everything I needed to know to finish my project here and being so kind and friendly at the same time.

# References


[1] ROOT Users guide 5.26 (`http://root.cern.ch/download/doc/Users_Guide_5_26.pdf`)

[2] Monte Carlo generators in ATLAS software, Journal of Physics: Conference Series 219 (2010) 032001 *C.Ay, A. Buckley, J. Butterworth, J. Ferland, I. Hinchliffe, O. Jinnouchi, J. Katzy, B. Kersevan, E. Lobodzinska, J. Monk, Z. Qin, V. Savinov, J. Schumacher*

[3] The ATLAS Data Model (`http://indico.cern.ch/event/66209/session/0/contribution/1/material/slides/1.pdf`) *Peter van Gemmeren*

[4] `http://atlas-proj-computing-tdr.web.cern.ch/atlas-proj-computing-tdr/Html/Computing-TDR-21.htm#pgfld-1019542`

[5] PhD. Thesis - Measurement of the $Z$ Boson production with the ATLAS Experiment at the LHC *Mikhail Karnevskiy*