SLAC 2016 data analysis: Documentation and explanations

Kim Siang Khaw
University of Washington

May 12, 2016

1 Introduction

Prerequisites for this tutorial are a basic understand of what the Muon g-2 experiment and PbF_2 calorimeters are, some knowledge about the electromagnetic (EM) shower and the ROOT data analysis framework. You can follow the exercise sheet step by step which guides you to the data analysis using the reconstructed electron EM shower clusters. Some analyses may require the use of reconstructed crystal hit, which is the basic object of forming a cluster. Advanced users may use the FNAL's *art* framework for the data analysis.

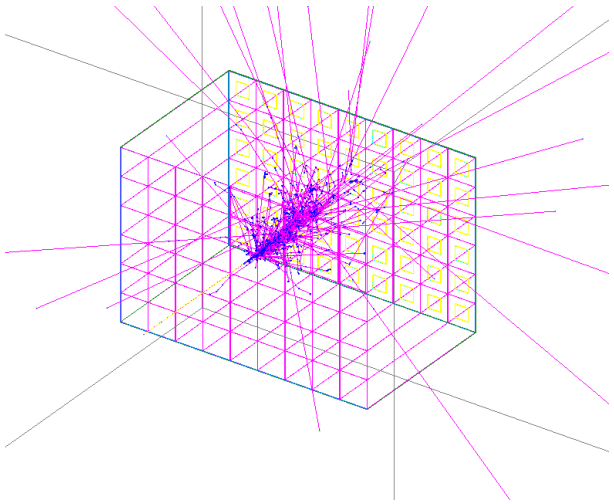

Figure 1: *Illustration of the electromagnetic shower of an electron injected from the left to the right, in a* 9×6 *array* PbF_2 *calorimeter. Cherenkov lights created by the charged shower particles are collected by the Silicon Photomultipliers (SiPMs) glued to the end of the* PbF_2 *crystals.*

Physics analysis of this test beam has several components that are common to the full Muon g-2 experiment. The physics objects that will be used for most of the analyses are the crystal hits and the clusters. These objects are reconstructed from the digitized waveform, after going through steps like pulsing fitting, energy calibration, gain correction and hit clustering. During the first week of the test beam, all these reconstruction steps will be refined based on the data taken and these will be taken care of by those who are familiar

with the *art* framework. The main focus of this documentation is on the analysis using high level physics objects like the crystal hit and the cluster.

Several studies that will be covered for the data analysis (non-exhaustive list) are

- energy resolution of the calorimeter
- position and angular resolution of the calorimeter
- degeneracy of the position and angular information of the calorimeter
- stability of gain monitoring system
- pile up separation for multi-electron events

Further information about how the data will be processed are summarized in Appendix A.

2 Data samples

2.1 File naming

The midas run number is preserved for each art/ROOT and ROOT files created from it.

Table 1: *Data samples and descriptions.*

filename	type
run00xxx_unpacked.root	data ready for art-based analysis
run00xxx_analysis.root	data ready for standalone C++ analysis

2.2 Location of the files

For the following steps, you have to be connected to the local Wifi network `g2SLAC`. All the art/ROOT and ROOT files are stored in the 2×3TB HDD in the `g2analyzer` machine. Use the following command to copy over the data files that you want to analyze:

```
scp -r g2muon@g2analyzer:/data/root_analysis/run00xxx_analysis.root
```

Or you can use `rsync` if you have enough space in your hard drive

```
rsync -avurt g2muon@g2analyzer:/data/root_analysis/ /your/folder/
```

3 Data format and structure for the ROOT tree

The data samples for this tutorial are stored in ROOT trees. The tree contains a collection of variables (called branches) which are filled once per event (could be 1 or more electrons). The list of variables along with their data type and further explanations are given in the following.

For studies using higher level objects

- `NCluster` (int): number of clusters in the event.
- `Cluster_X` (double): local x-position of the cluster in the event.
- `Cluster_Y` (double): local y-position of the cluster in the event.
- `Cluster_Z` (double): local z-position of the cluster in the event.
- `Cluster_E` (double): energy (MeV) of the cluster in the event.
- `Cluster_T` (double): time (ns) of the cluster in the event.
- `NXtalHit` (int): number of crystal hits in the event.
- `XtalHit_X` (double): local x-position of the crystal hit in the event.
- `XtalHit_Y` (double): local y-position of the crystal hit in the event.
- `XtalHit_Z` (double): local z-position of the crystal hit in the event.
- `XtalHit_E` (double): energy (MeV) of the crystal hit in the event.
- `XtalHit_T` (double): time (ns) of the crystal hit in the event.
- `XtalHit_XtalNum` (int): crystal number in a calorimeter of the crystal hit in the event.

For studies using lower level objects

- `Cluster_X` (double): local x-position of the cluster in the event.
- `Cluster_Y` (double): local y-position of the cluster in the event.
- `Cluster_Z` (double): local z-position of the cluster in the event.
- `Cluster_E` (double): energy (MeV) of the cluster in the event.

4 Standalone C++ Analysis framework

The package `SLAC2016Ana.tar.gz` contains an example `C++` framework to help you getting started. Unpack the example with `tar xvzf SLAC2016Ana.tar.gz` to your work folder. You can build your additional analysis code on top of this example or write a new one based on it. The example is already running but it's not doing much yet. You can compile the analysis code by first sourcing your ROOT environment (e.g. `source /usr/local/bin/thisroot.sh`) and then followed by executing the command `make`.

This will read the necessary ingredients for compilation from the `Makefile` in the same directory. Don't have to worry much about this file at the moment unless you want to add in more classes to the analysis code. The point is that it creates an executable named `ana`. You can then execute the program by the command `./ana input.script` where `input.script` includes a path to the root file that you want to analyze (e.g. `./test.root`).

A description of the individual components of the example are given in the following list. Indicated are also the places where you should start adding your own code:

- `main.cxx`: This is the first starting point. It contains the `main()` function which is necessary for any C++ program. The first step is to create instances of `MyAna()` class which is implemented in the files `MyAna.h` and `MyAna.C` (explained in the next items). The `TChain` represents the ROOT tree discussed in section 3. The files which should be read from disk are specified in the function `Add(filename)`. The tree is then read and processed by the `MyAna()` class which takes the `TChain` as argument. The real work is then done in the `Loop()` function of the `MyAna()` class which is discussed in the next two items.

- `MyAna.h`: Definition of the class `MyAna`, which inherits from the `TTree::MakeClass`. It declares variables and ROOT objects that will be used or stored in your analysis. Several basic functions that are common among event-based particle physics analysis like `initialize()`, `clear()`, `execute()` and`finalize()` are declared here.

- `MyAna.C`: The main function which is called automatically which processing the ROOT trees are `Loop()`. The `Loop()` function is called only once per run. In the `Loop()` function, `initialize()` is called at the beginning of the analysis run, `clear()` and `execute()` is called every event, and `finalize()` at the end of the analysis run.

- `t1.h`: Header file for the class `t1` created using `TTree::MakeClass`.

- `t1.C`: Source file for the class `t1` created using `TTree::MakeClass`. The class `Loop()` is used by `MyAna` to loop through each `TBranch`.

- `PlotAll.C`: A ROOT macro which can be used for automatic plotting of a set of histograms which are stored in a ROOT file. Please read the header of the file on how to use it.

5 ROOT-based offline event display

A ROOT-based offline event display is also developed to increase the user friendliness of the data analysis. As shown in Figure 2, this GUI interface allows the user to inspect the fit results of the template fit algorithm by overlaying it with the island samples. More functionality will be added in the coming days.

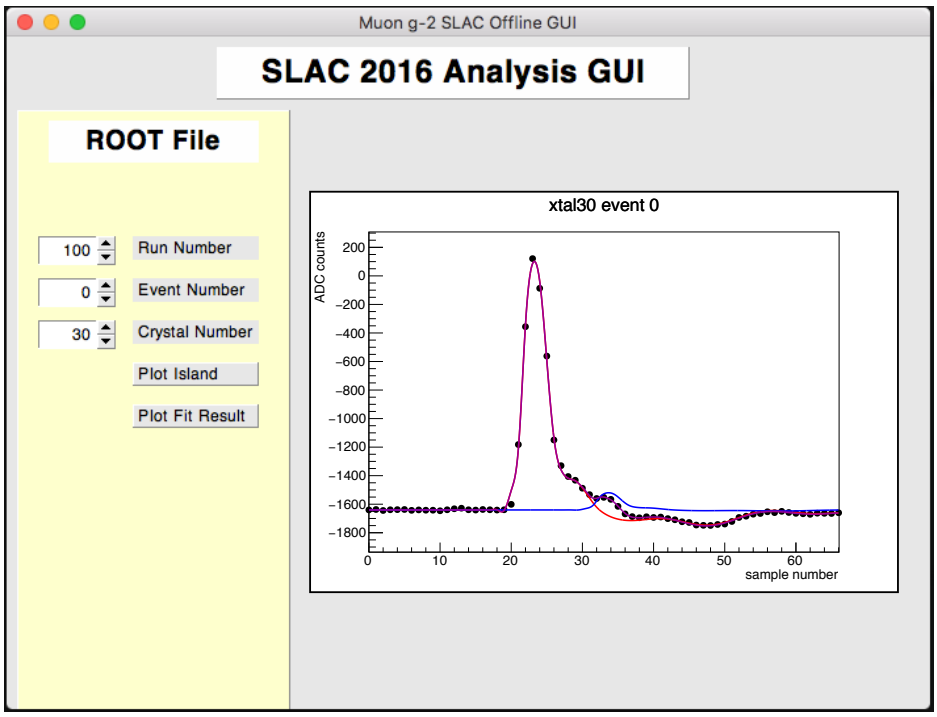

Figure 2: *SLAC Offline analysis GUI. This is a very preminary prototype.*

A Data analysis framework

The data analysis of this test beam has several components. First we need to convert the raw data stored in a MIDAS file (`.mid` or `.mid.gz`) to *art* data products stored in a ROOT file. This is handled using *art* framework's modules and is doing nothing more than storing `16-bit` or `32-bit` word into `vectors`. Next we unpack these `vectors` and give them contexts based on the header information stored within the `vectors`. As this step, all the information are stored as data products you are probably familiar with: `RiderArtRecord`, `IslandArtRecord`, etc.

B Header information

This section provides the information regarding user interface to the event, AMC13 and rider header information. All the information are stored in the art/ROOT files and standalone ROOT files with the TBranch structures in the following sections.

B.1 Terminology

Several technical terms defined in this document. A run consists of multiple events. There is a maximum of 12 AMC slots per calorimeter. Each AMC slot has a Rider digitizer, and each Rider has 5 input channels.

B.2 RunHeader Class Reference

Run header appears once for each run.

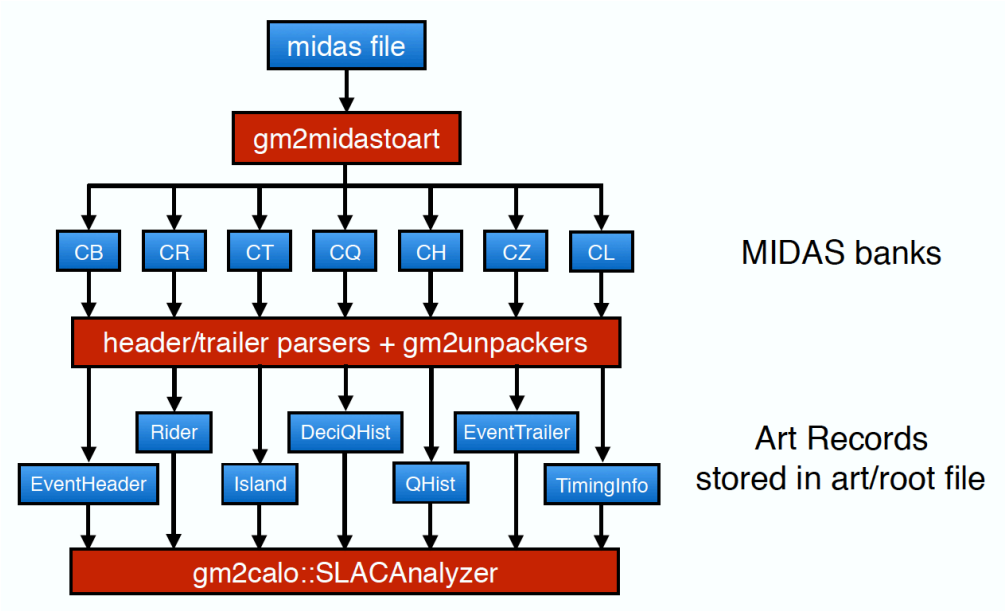

Figure 3: *Offline framework for SLAC experimental data.*

TLeaf members

Below are the member variables that appear under TBranch RunHeader.

- ULong_t **RunNum**
- ULong_t **NumOfEvent**
- ULong_t **NumOfAMC13Trig**
- ULong_t **NumOfRiderTrig**

B.3 EventHeader Class Reference

Event header appears once for each event.

TLeaf members

Below are the member variables that appear under TBranch EventHeader.

- ULong_t **EventNum**
- ULong_t **AMC13TrigNum**
- UChar_t **EventType**
- UChar_t **NumOfAMC**

- UChar_t **NumOfRiderChannel**
- struct **AMCHeader**
  - UChar_t **AMCSlotNum**
  - ULong_t **RiderSerialNum**
  - ULong_t **AMCCounter**
  - ULong_t **RiderTrigNum**

B.4 RiderChannelHeader Class Reference

Rider channel header appears once for each rider channel.

TLeaf members

Below are the member variables that appear under TBranch RiderChannelHeader.

- struct **RiderChannelHeader**
  - UChar_t **AMCSlotNum**
  - UChar_t **RiderChannelNum**
  - UChar_t **FillType**
  - ULong_t **WaveformIndex**
  - ULong_t **WaveformCount**
  - ULong_t **TriggerNum**
  - UInt_t **WaveformLength**

C Header and trailer formats

This section compiles all the available header data formats. Please refer to http://gm2-docdb.fnal.gov:8080/cgi-bin/ShowDocument?docid=3409 for more details.

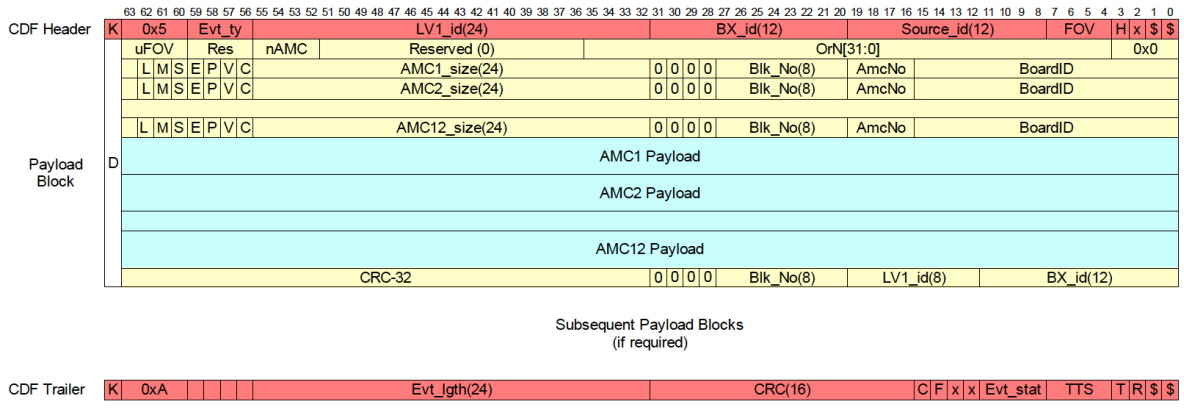

Figure 4: *AMC13 to DAQ data format.*

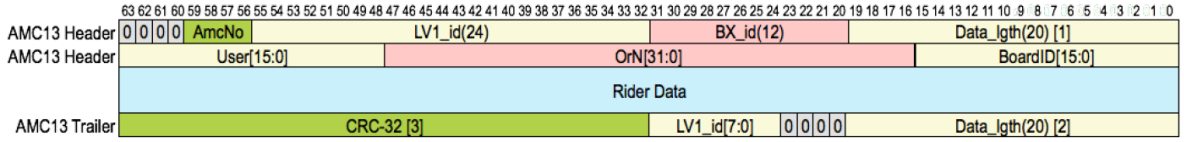

Figure 5: *Rider to AMC13 data format.*

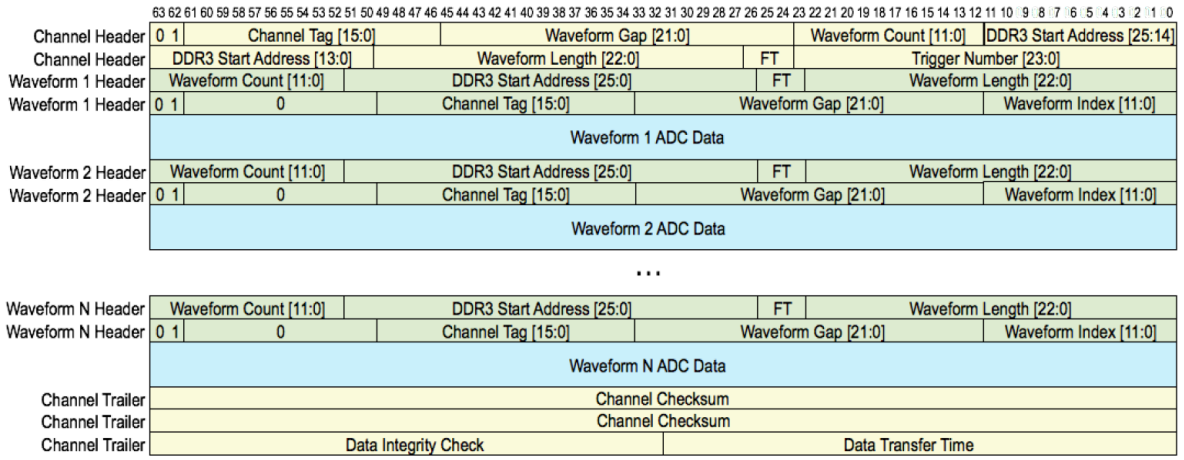

Figure 6: *Rider Channel data format.*

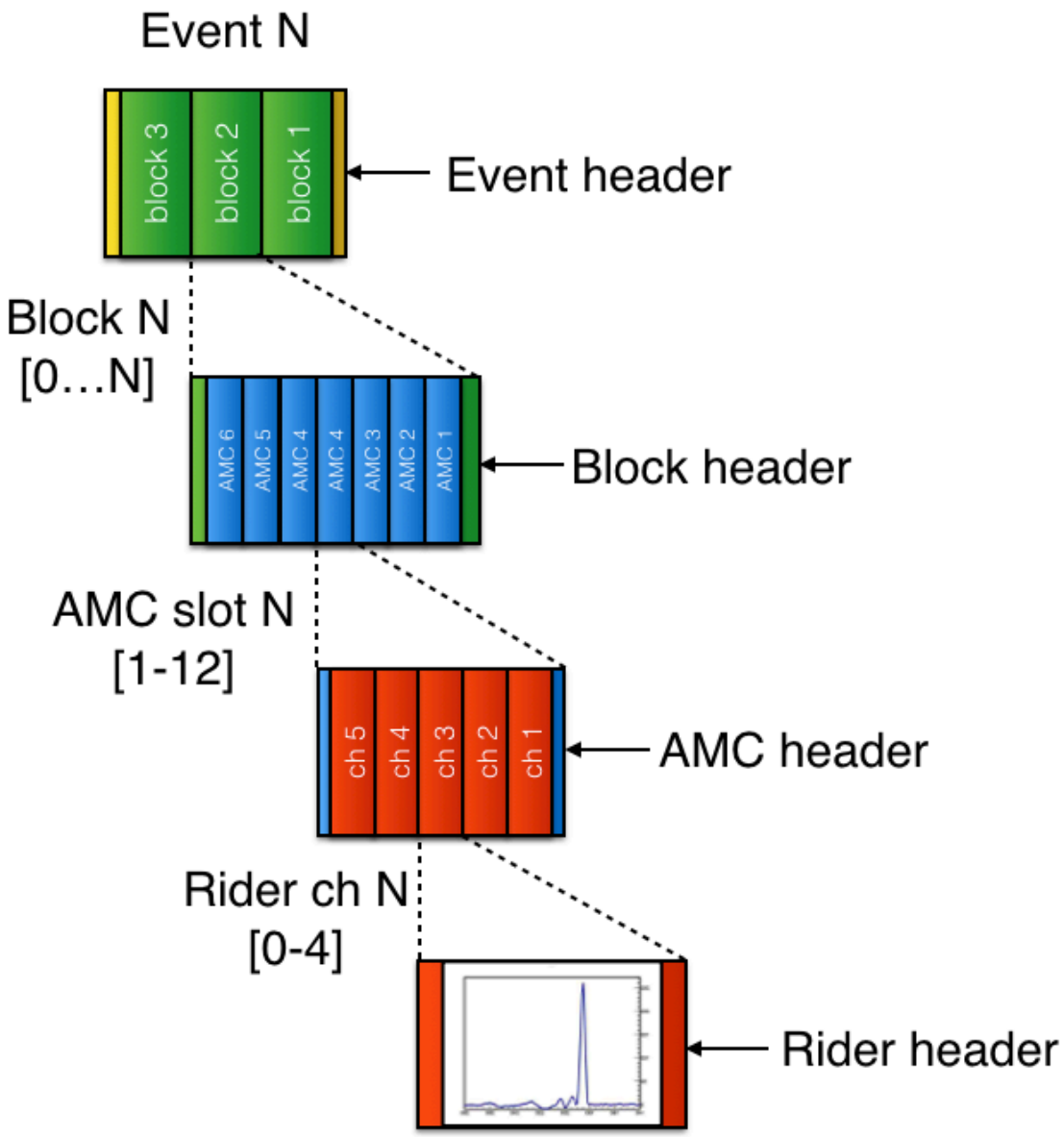

Figure 7: *Per event data format*