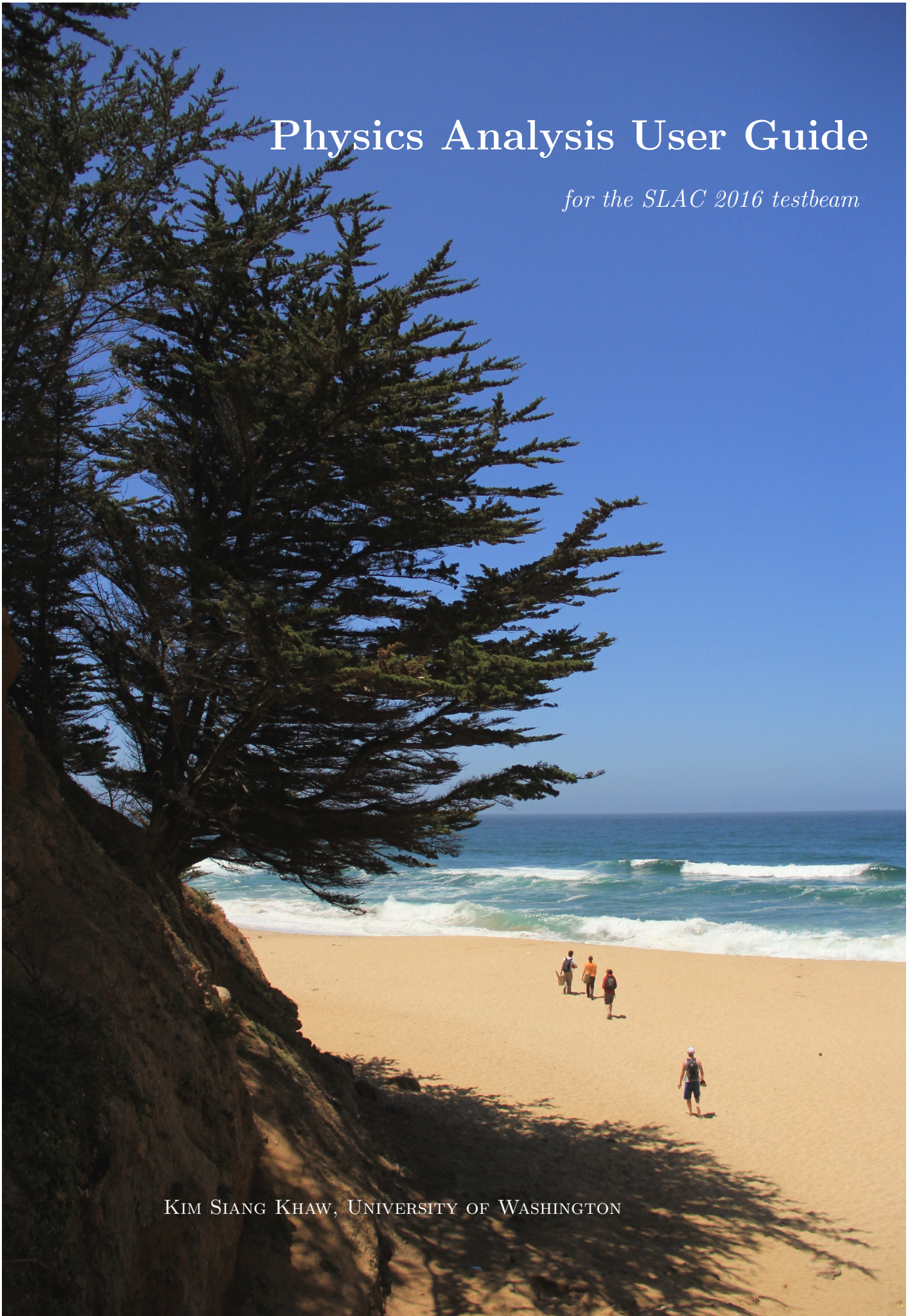
Physics Analysis User Guide

for the SLAC 2016 testbeam



Kim Siang Khaw, University of Washington

Abstract

This is a documentation for the SLAC2016 data analysis.

Contents

Abstract ii

Contents iii

List of Figures v

List of Tables vii

Acronyms ix

1 Introduction 1

2 T-536 program 3

3 Runlog 5

4 Data analysis framework 7
4.1 Offline framework for the SLAC test beam 7

5 Get started 9
5.1 Installation 9
5.1.1 Setting up the environment 9
5.1.2 Checkng out unpacking and reconstruction packages 10
5.1.3 Data unpacking 12
5.1.4 Reconstruction 12

6 Standalone C++ Analysis framework 13
6.1 Data format and structure for the ROOT tree 14

7 ROOT-based offline event display 19

8 Napoli DAQ 21

9 Fiber harp **23**

A Data location **25**
A.1 Data samples 25
A.1.1 File naming 25
A.1.2 Location of the files 25

B T-536 program **27**

C Daq information **37**
C.1 DAQ 37
C.1.1 Header information 37
C.1.2 Header and trailer formats 37

Bibliography **41**

List of Figures

1.1 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. 1

4.1 Offline framework for SLAC experimental data. 7

7.1 SLAC Offline analysis GUI. This is a very preminary prototype. 19

C.1 AMC13 to DAQ data format. 37
C.2 Rider to AMC13 data format. 38
C.3 Rider Channel data format. 38
C.4 Per event data format 39

List of Tables

A.1 Data samples and descriptions. The ROOT full data includes raw waveforms, chopped islands and processed data like fit results and crystal hits. The ROOT analysis data has only processed data. 25

Acronyms

MIDAS	Maximum Integrated Data Acquisition System
TRUIMF	Canada's national laboratory for particle and nuclear physics and accelerator-based science
PSI	Paul Scherrer Institute

Chapter 1

Introduction

Prerequisites for this tutorial are a basic understanding of what the Muon g-2 experiment and PbF_2 calorimeters [1] are, some knowledge about the electromagnetic (EM) shower and the ROOT data analysis framework [2]. 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 are encouraged to use the FNAL's *art* framework for the data analysis.

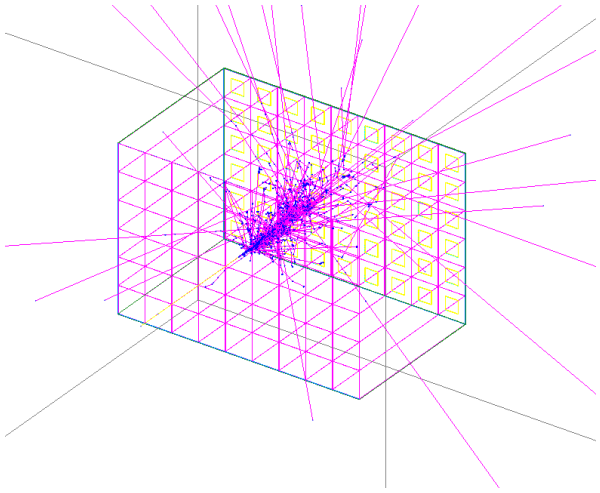


Figure 1.1: *Illustration of the electromagnetic shower of an electron injected from the left to the right, in a* 9×6 *array* PbF_2 *calorimeter. Cherenkov lights created by the charged shower particles are collected by the Silicon Photomultipliers (SiPMs) glued to the end of the* PbF_2 *crystals.*

Data analysis of this test beam has several components that are common with the Muon g-2 experiment data analysis framework [3]. The physics objects that will be used for most of the analyses are the crystal hits and the hit clusters. These objects are reconstructed

from the digitized waveform, after going through steps like pulsing fitting, energy calibration, gain correction, time 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 physics analysis using high level physics objects like the crystal hit and the cluster, using a C++/ROOT standalone framework. The user can proceed to more sophisticated analysis once he/she is familiar with the tools. The standalone framework can also be used as a stepping stone to develop analysis algorithm before migrating it to the *art* framework.

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 and analyzed are summarized in Appendix A (to be updated).

Chapter 2

T-536 program

This chapter summarizes the program we have planned during the SLAC run. For full program, please refer to Appendix B.

Chapter 3

Runlog

This chapter summarizes the run log in the mysql database written out by the Maximum Integrated Data Acquisition System (MIDAS) DAQ.

Chapter 4

Data analysis framework

4.1 Offline framework for the SLAC test beam

MIDAS DAQ system is developed at Paul Scherrer Institute (PSI) and Canada's national laboratory for particle and nuclear physics and accelerator-based science (TRUIMF).

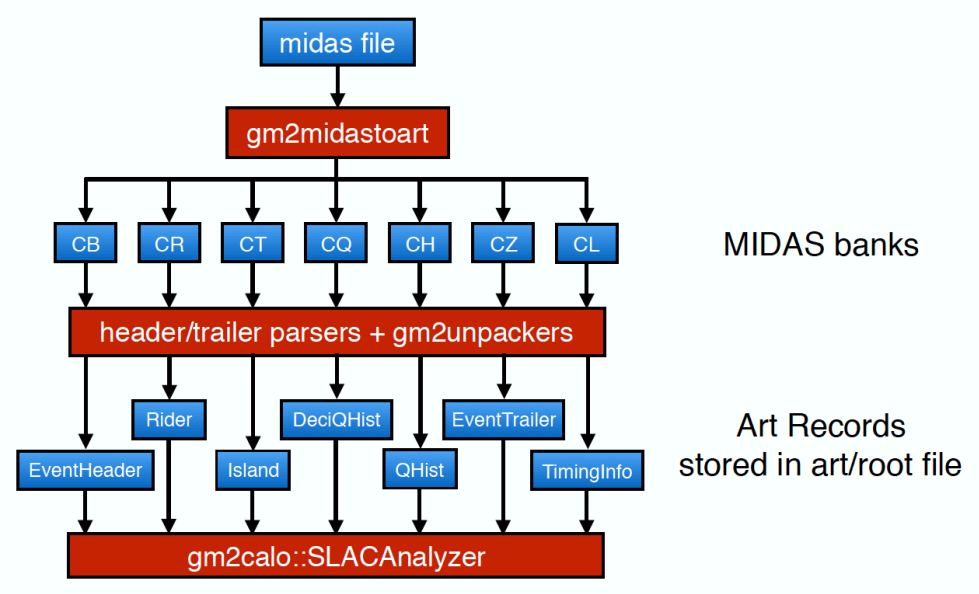


Figure 4.1: *Offline framework for SLAC experimental data.*

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.

Chapter 5

Get started

5.1 Installation

This section is based on the official Offline Computing and Software Manual http://gm2-docdb.fnal.gov:8080/cgi-bin/ShowDocument?docid=1825. Only important steps are reproduced here.

5.1.1 Setting up the environment

First of all you need to login to the Fermilab virtual machine (`gm2gpvm02-04`). Once you have logged in, you need to choose a release for the Muon g-2 softwares (libraries, executables) every time you login or you can put it in your `.profile` on `gm2gpvm`. Do the following:

```
source /grid/fermiapp/gm2/setup
```

Alternatively if you want to run codes on your laptop, you can use cvmfs as mentioned in the documentation.

```
source /cvmfs/gm2.opensciencegrid.org/prod/g-2/setup
```

If successful, you will get

```
Using g-2 release repository at /cvmfs/gm2.opensciencegrid.org
g-2 software

--> To list gm2 releases, type
ups list -aK+ gm2
```

```
--> To use the latest release, do
setup gm2 v6_04_00 -q prof

For more information, see
https://cdcvs.fnal.gov/redmine/projects/g-2/wiki/ReleaseInformation.
```

Then follow the instruction and use the latest release of `gm2` (`v6_04_00` at the moment of writing this user guide). Next to do is to go to a folder of your choice (usually `~/work`) and then create a new development place.

```
cd ~/work
mkdir SLACDev
```

Now go into the newly created folder `SLACDev` and initialize it as a new development area

```
cd SLACDev/
mrb newDev
```

Follow the instruction to source the localproducts settings

```
source localProducts_gm2_v6_04_00_prof/setup
```

Now we are ready to install new packages.

5.1.2 Checkng out unpacking and reconstruction packages

First we need to go into the `srcs/` folder to checkout the unpacking and reconstruction packages. Packages related to the data unpacking are `gm2midas`, `gm2midastoart` and `gm2unpackers` whereas those related to the reconstruction are `gmcalo`. `gm2dataproducts` holds all the data structures for both.

Checkout (`git`) `gm2calo` package using simplified command `mrb g`.

```
cd srcs/
mrb g gm2midas; mrb g gm2midastoart; mrb g gm2unpackers;
mrb g gm2calo; mrb g gm2dataproducts
```

For the package `gm2midas`, you need to use the master branch,

```
cd gm2midas
git checkout master
```

and build both `midas` and `rome`:

```
cd midas
MIDASSYS=‘pwd‘
MIDASSYS=‘pwd‘ >> ~/.bash_profile
make; make install
cd ../rome
ROMESYS=‘pwd‘
ROMESYS=‘pwd‘ >> ~/.bash_profile
make; make install
```

Then go to `gm2midastoart` to build the midas file library provided by ROME,

```
cd ../../gm2midastoart
git flow feature track SLAC2016
cd rome
make; make install
```

For the packages `gm2unpackers`, `gm2calo` and `gm2dataproducts` you need to use the SLAC2016 feature branch (you will be checking out develop branch if you do nothing)

```
cd ../../gm2calo (gm2unpackers, gm2dataproducts)
git flow feature track SLAC2016
cd ..
```

Execute `'mrb uc'` to update the CMakeLists to include all the packages:

```
mrb uc
```

Setup `ninja` (a small build system with focus on speed) with

```
setup ninja v1_5_3a
```

Read more about it at https://ninja-build.org/. Then start to build the packages with

```
. mrb s
mrb b --generator ninja
```

If the build was successful (it takes about 5 minutes), you will see the following:

```
------------------------------------
INFO: Stage build successful.
------------------------------------
```

Now set an alias for the ninja (because it is fast! Not for the first build though.) build and put it in your `.bash_profile`

```
alias ninja='pushd $MRB_BUILDDIR; ninja; popd'
```

Now you are ready to unpack the data, reconstruct the data and analyze the data.

5.1.3 Data unpacking

`fcl` files for the data unpacking are location in the `gm2unpackers/fcl/` folder.

5.1.4 Reconstruction

`fcl` files for the data unpacking are location in the `gm2calo/fcl/` folder.

Chapter 6

Standalone C++ Analysis framework

The package `SLAC2016Ana` contains an example C++ framework to help you getting started. You can get it from the github.com by `git clone https://github.com/kimsiang/SLAC2016`. 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.

6.1 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

- `FitResult_EventNum` (`vector<int>`): event number of this fit result
- `FitResult_CaloNum` (`vector<int>`): calorimeter number of this fit result
- `FitResult_XtalNum` (`vector<int>`): crystal number of this fit result
- `FitResult_IslandNum` (`vector<int>`): island number of this fit result
- `FitResult_UtcaSlotNum` (`vector<int>`): utca slot number of this fit result
- `FitResult_ChannelNum` (`vector<int>`): rider channel number of this fit result
- `FitResult_Energy` (`vector<double>`): energy (number of photons) of this fit result
- `FitResult_Time` (`vector<double>`): time (clock tick of 800 MHz) of this fit result within the fill/event
- `FitResult_Pedestal` (`vector<double>`): pedestal of this fit result

- `FitResult_Chi2` (`vector<double>`): Chi squared of this fit result
- `FitResult_ClockCounter` (`vector<long long>`): time stamp (clock tick of 40 MHz) of this fit result from Rider header information
- `XtalHit_EventNum` (`vector<int>`): event number of this crystal hit
- `XtalHit_CaloNum` (`vector<int>`): calorimeter number of this crystal hit
- `XtalHit_XtalNum` (`vector<int>`): crystal number of this crystal hit
- `XtalHit_IslandNum` (`vector<int>`): island number of this crystal hit
- `XtalHit_UtcaSlotNum` (`vector<int>`): utca slot number of this crystal hit
- `XtalHit_ChannelNum` (`vector<int>`): rider channel number of this crystal hit
- `XtalHit_Energy` (`vector<double>`): energy (number of photons) of this crystal hit
- `XtalHit_Time` (`vector<double>`): time (clock tick of 800 MHz) of this crystal hit within the fill/event
- `XtalHit_ClockCounter` (`vector<long long>`): time stamp (clock tick of 40 MHz) of this crystal hit from Rider header information
- `Cluster_EventNum` (`vector<int>`): event number of this cluster
- `Cluster_CaloNum` (`vector<int>`): calorimeter number of this cluster
- `Cluster_IslandNum` (`vector<int>`): island number of this cluster
- `Cluster_Energy` (`vector<double>`): energy (number of photons) of this cluster
- `Cluster_Time` (`vector<double>`): time (clock tick) of this cluster
- `Cluster_X` (`vector<double>`): local x-position of this cluster (logarithmic-weighted)
- `Cluster_Y` (`vector<double>`): local x-position of this cluster (logarithmic-weighted)
- `Italiano_EventNum` (`vector<int>`): event number of this analyzed laser crate waveform
- `Italiano_CaloNum` (`vector<int>`): calorimeter number of this analyzed laser crate waveform
- `Italiano_XtalNum` (`vector<int>`): crystal number of this analyzed laser crate waveform

- `Italiano_IslandNum` (`vector<int>`): island number of this analyzed laser crate waveform
- `Italiano_UtcaSlotNum` (`vector<int>`): utca slot number of this analyzed laser crate waveform
- `Italiano_ChannelNum` (`vector<int>`): rider channel number of this analyzed laser crate waveform
- `Italiano_Amplitude` (`vector<double>`): amplitude (ADC samples) of this analyzed laser crate waveform
- `Italiano_Time` (`vector<double>`): time (clock tick of 800 MHz) of this analyzed laser crate waveform within the fill/event
- `Italiano_Pedestal` (`vector<double>`): pedestal of this analyzed laser crate waveform
- `Italiano_Area` (`vector<double>`): Area of this analyzed laser crate waveform
- `Italiano_ClockCounter` (`vector<long long>`): time stamp (clock tick of 40 MHz) of this analyzed laser crate waveform from Rider header information

For studies using lower level objects

Blow are the Tleaves of the `islandTree`.

- `RunNum` (`int`): run number of this island
- `EventNum` (`int`): event number of this island
- `FillType` (`int`): fill type of this island (1 is muon fill, 2 is laser fill)
- `TriggerNum` (`int`): trigger number of this island
- `CaloNum` (`int`): calorimeter number of this island
- `XtalNum` (`int`): crystal number of this island
- `IslandNum` (`int`): island number of this island
- `UtcaSlotNum` (`int`): utca slot number of this island
- `ChannelNum` (`int`): rider channel number of this island
- `Length` (`int`): number of sample of this island

- `Time` (`int`): time (clock tick of 800 MHz) of the first sample of this island within the fill/event
- `ClockCounter` (`long`): time stamp (clock tick of 40 MHz) of this island from Rider header information
- `Trace` (`vector<short>`): ADC samples of this island

Chapter 7

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 7.1, 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.

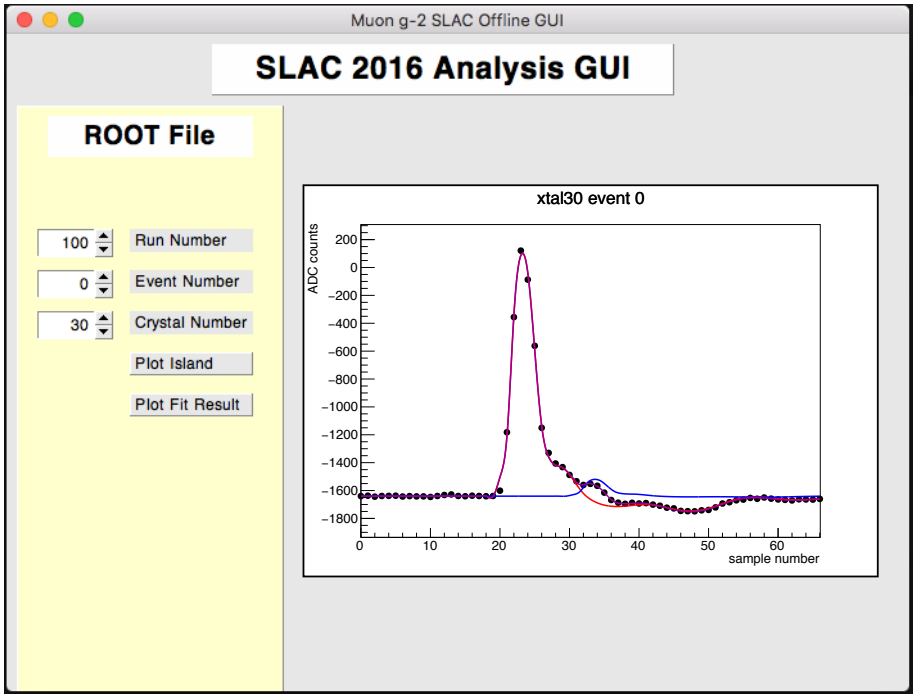


Figure 7.1: *SLAC Offline analysis GUI. This is a very preminary prototype.*

Chapter 8

Napoli DAQ

In this chapter, the Napoli DAQ will be explained. The codes for the plots are in https://github.com/gm2-it/test-beam-slac-jun16/tree/master/NapoliDaq/analyzer.

Chapter 9

Fiber harp

Appendix A

Data location

This chapter summarizes the location of the files.

A.1 Data samples

This section explains the naming and the locations of the data files (Local location will be obsolete once we move everything to the PNFS disk at FNAL).

A.1.1 File naming

All the created art/ROOT and ROOT files will have the same run number as the MIDAS they are extracted from.

Table A.1: *Data samples and descriptions. The ROOT full data includes raw waveforms, chopped islands and processed data like fit results and crystal hits. The ROOT analysis data has only processed data.*

filename	type
gm2slac_run0xxxx.art	data ready for art-based analysis
gm2slac_run0xxxx_raw.root	full data for standalone C++ analysis
gm2slac_run0xxxx.root	analysis data for standalone C++ analysis

A.1.2 Location of the files

For the following steps, you have to be connected to the local Wifi network `T536-local`. All the art/ROOT and ROOT files are stored in the 2×3TB HDD in the `g2analysis` machine. Use the following command to copy over the data files that you want to analyze:

```
scp -r g2muon@g2analysis:/data1/slac2016/analysis/gm2slac_run0xxxx.root .
scp -r g2muon@g2analysis:/data2/slac2016/art/gm2slac_run0xxxx.art .
```

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

```
rsync -avurt g2muon@g2analysis:/data1/slac2016/analysis/ YOUR_LOCAL_DISK/
rsync -avurt g2muon@g2analysis:/data2/slac2016/art/ YOUR_LOCAL_DISK/
```

Or a better way is to use `sshfs` to mount the disk onto your laptop:

```
sshfs g2muon@g2analysis:/data1/slac2016/analysis YOUR_LOCAL_DISK
sshfs g2muon@g2analysis:/data2/slac2016/art YOUR_LOCAL_DISK
```

Appendix B

T-536 program

This chapter summarizes the program we have planned during the SLAC run.

Study	Comment / Instruction / Run Numbers
Vertical Sweep	1649 - 1653
Calibration Sequnce	Runs: 1665 - 1672
Template xtal 24	Runs: 1673 - ... one hour
Test PIX image of beam	(failed so far)
crystal centers in the cluster of 4x4 crystals:	2000 events per run
34, 33, 32, 31 25, 24, 23, 22 16,15,14,13	use relPos buttons to move red means done
high statistics seg 24	1 hour
Fine position scan of segment 24	for y in 0, and 10 for x from -32 to 32 in steps of 4 (64 mm scanned out of 64 for y=0) Current position x=277, y=95, Xtal 23 3000 events per run total run time: 6 hours
Cross of 4 segments	Intersection of segment 24, 25, 33, 34 Intersection of segment 25, 26, 34, 35
Edge scan outward of segment 26	For y=0, 6 points from center to calo edge For y=10, 6 points from center to calo edge Remain to be done: x={332.6,330.1,327.6}, y=95
Crack scan	Crack 24-25 ; 25-26 ; fine for 24-25-33-34

Energy scan at 3 points for each 3.5 GeV 4.0 GeV 4.5 GeV 5.0 GeV 2.5 GeV	4 X,Y Locations at each energy (277,105) (291.8,105) (304.4,105) (291.8,92.4) (279.2,105) (291.8,105) (304.4,105) (291.8,92.4) (279.2105) (291.8,105) (304.4,105) (291.8,92.4) (279.2,105) (291.8,105) (304.4,105) (291.8,92.4) (279.2,105) (291.8,105) (304.4,105) (291.8,92.4) (279.2,105)
Edge scan outward of segment 26	For y=95, x=347.45 (outside); 344.95 (outside); 342.45 ; 339.95 ; 337.45; 334.95; 332.45; 329.95 Repeat for y=105, x=347.45 (outside); 344.95 (outside); 342.45 ; 339.95 ; 337.45; 334.95; 332.45; 329.95
Angle Scan at 5 degrees	3000 events per step Y = 105; X = [284, 288, 292, 296, 300, 304, 308, 312, 316, 320, 324, 328, 332, 336, 340, 344, 348, 352, 356, 360] 3000 events per step Y = 95; X = [284, 288, 292, 296, 300, 304, 308, 312, 316, 320, 324, 328, 332, 336, 340, 344, 348, 352, 356, 360]
Angle Scan at 10 degrees	3000 events per step Y = 105; X = [286, 290, 294, 298, 302, 306, 310, 314, 318, 322, 326, 330, 334, 338, 342, 346, 350, 354, 358, 362] 3000 events per step Y = 95; X = [286, 290, 294, 298, 302, 306, 310, 314, 318, 322, 326, 330, 334, 338, 342, 346, 350, 354, 358, 362] then a long run at Y = 95, X = 306 till 9AM

Angle Scan at 15 degrees	3000 events per step Y = 105; X = [294, 298, 302, 306, 310, 314, 318, 322, 326, 330, 334, 338, 342, 346, 350, 354, 358, 362, 366, 370] 3000 events per step Y = 95; X = [294, 298, 302, 306, 310, 314, 318, 322, 326, 330, 334, 338, 342, 346, 350, 354, 358, 362, 366, 370] then a long run at Y = 95, X = 310 till 9PM with filter wheel calibration in parallel
Moving Flight Simulator Tests (1)	FSrate=32; FWstate=6; 1/2 mode; T0 = -110950 (start of exp sequence) scan the first 20 usec, 3000 events per run trigger delay -120950, -118950, -116950, -114950, -112950, -110950, -120950, -122950, -124950, -126950, -128950, -130950, scan the first 100 usec, 3000 events per run trigger delay -130950, -140950, -150950, -160950, -170950, -190950, -210950 60 min runs Delay = 15 us, 30 us, 60 us, 120 us, 240 us, 480 us; Check statistics along way for run length precision in ratio of on/off

Moving Flight Simulator Tests (2)	FSrate=32; FWstate=5; 1/2 mode; T0 = -110950 (start of exp sequence) scan the first 20 usec, 3000 events per run trigger delay -120950, -118950, -116950, -114950, -112950, -110950, -120950, -122950, -124950, -126950, -128950, -130950, scan the first 100 usec, 3000 events per run trigger delay -130950, -140950, -150950, -160950, -170950, -190950, -210950
Moving Flight Simulator Tests (3)	FSrate=32; FWstate=1; 1/2 mode; T0 = -110950 (start of exp sequence) scan the first 20 usec, 6000 events per run trigger delay -120950, -116950, -112950 (stopped), -124950, -130950, -140950, -160950
Fine vertical scan seg 24	3000 events per run at X = 284, and Y = [85, 89, 93, 97, 101, 105, 109, 113, 117, 121, 125]
Horizontal scan seg 24	3000 events per run at Y = 105 and X = 280, 284, 288
Front face scan	widen island for template gen (100 pre, 200 post) 4500 events per segment 26, 25, 24, 23, 22, 21, 20, 19, 18, 9, 10, 11, 12, 13, 14, 15, 16, 17, 35, 34, 33, 32, 31, 30, 29, 28, 27, 36, 37, 38, 39, 40, 41, 42, 43, 44, 53, 52, 51, 50, 49, 48, 47, 46, 45, 0, 1, 2, 3, 4, 5, 6, 7, 8
undo wide islands for templates back to 8 pre-trigger, and 24 post-trigger	

Flight sim test, FW#1, 96 shots per fill	FSrate=96; FWstate=1; 1/2 mode; 9000 events per run trigger delay -120950, -118950, -116950, -114950, -112950, -122950, -124950, -126950, -128950, -130950,
redo ping seg 8, 3	switch laser to 100kHz fixed widen islands for template generation collect 4500 events undo wide islands
low QE run	be creative
long run	25 deg, two diff positions, same beam, 6 hours each, filter wheel calibration before, after and during
position scan for 25 deg	9000 events per point Y = 105, X = 284.0, 288.0, 292.0, 296.0, 300.0, 304.0, 308.0, 312.0, 316.0, 320.0, 324.0, 328.0, 332.0, 336.0, 340.0, 344.0, 348.0, 352.0, 356.0, 360.0, 364.0, 368.0, 372.0, 376.0, 380.0 9000 events per point Y = 95, X = 284.0, 288.0, 292.0, 296.0, 300.0, 304.0, 308.0, 312.0, 316.0, 320.0, 324.0, 328.0, 332.0, 336.0, 340.0, 344.0, 348.0, 352.0, 356.0, 360.0, 364.0, 368.0, 372.0, 376.0, 380.0

horizontal scan of fiber harp	3000 events per run Y = 114, 110, 106, 102, 98 X = 240, 244, 248, 252, 256, 260, 264, 268, 272, 276, 280, 284, 288, 292, 296 300, 304, 308, 312, 316, 320, 324, 328, 332, 336, 340, 344, 348, 352, 356, 360
horizontal scan of fiber harp in calibration position, vertically centered in beam	3000 events per run Y = 105 X = 240, 244, 248, 252, 256, 260, 264, 268, 272, 276, 280, 284, 288, 292, 296, 300, 304, 308, 312, 316, 320, 324, 328, 332, 336, 340, 344, 348, 352, 356, 360
scan of intensity and bias voltage	at Y = 105, X = 260 mm bias voltages = 65.5, 66.0, 66.5, 67.0, 67.5, 68.0, 68.5 VSL10 = 20, 15, 10, 5 mm C24_H = 1, 2, 4 mm 6000 events if SL10*C24_H <= 10 mmˆ2; otherwise 3000 events

horizontal scan of fiber harp in calibration position with narrow beam	3000 events per run Y = 105, X = 240, 244, 248, 252, 256, 260, 264, 268, 272, 276, 280, 284, 288, 292, 296, 300, 304, 308, 312, 316, 320, 324, 328, 332, 336, 340, 344, 348, 352, 356, 360, X = 242, 246, 250, 254, 258, 262, 266, 270, 274, 278, 282, 286, 290, 294, 298, 302, 306, 310, 314, 318, 322, 326, 330, 334, 338, 342, 346, 350, 354, 358
long run	1 hour = 36,000 events total 6 runs of 6000 events each (10 min) Y = 105, X = 260 SL10 = 20, C24 = 1.5
bias voltage scan	3000 events, Y = 105, X = 260, SL10 = 20, C24 = 1.5, bias voltages = 65.5, 66.0, 66.5, 67.0, 67.5, 68.0, 68.5 V

horizontal scan of fiber harp in calibration position, new beam width	3000 events per run SL10 = 20 mm, C24 = 1.5 mm, Y = 105, X = 240, 244, 248, 252, 256, 260, 264, 268, 272, 276, 280, 284, 288, 292, 296, 300, 304, 308, 312, 316, 320, 324, 328, 332, 336, 340, 344, 348, 352, 356, 360
scan of intensity and bias voltage at ideal position along fibers	at Y = 105, X = 276 mm bias voltages = 65.5, 66.5, 67.5, 68.5 V, C24_H = 1.5, 2, 4 mm SL10 = 20, 10, 5 mm, 3000 events
scan of intensity and bias voltage at ideal position along fibers	at Y = 105, X = 276 mm bias voltages = 66, 67, 68 V (at C24 = 2, SL10 = 20, do more points: 65.5, 66, 66.5, 67, 67.5, 68, 68.5 V) C24_H = 1.5, 2, 4 mm SL10 = 20, 10, 5 mm 3000 events
long run at ideal position along fibers	1 hour = 36,000 events total 6 runs of 6000 events each (10 min) Y = 105, X = 276 SL10 = 20, C24 = 2

long run with white paint on fiber ends	1 hour = 36,000 events total 6 runs of 6000 events each (10 min) Y = 105, X = 276 SL10 = 20, C24 = 2
horizontal scan of fiber harp in calibration position with white paint	3000 events per run SL10 = 20 mm, C24 = 2 mm, Y = 105, X = 264, 268, 272, 276, 280, 284, 288, 292, 296, 300, 304, 308, 312, 316, 320, 324, 328, 332, 336, 340, 344, 348, 352, 356
vertical scan of T0 counter	3000 events per run SL10 = 20 mm, C24 = 2 mm X = 288 Y = 65, 75, 85, 95, (105), 115, 125, 135, 145
laser before beam at different time separations	3000 events, Y = 105, X = 276, SL10 = 20, C24 = 2, deltaT = 10, 20, 30, 40, 50, 75, 100, 150, 300 ns

Appendix C

Daq information

This chapter summarizes the minimum information from the DAQ needed for the analysis.

C.1 DAQ

C.1.1 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.

C.1.2 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.

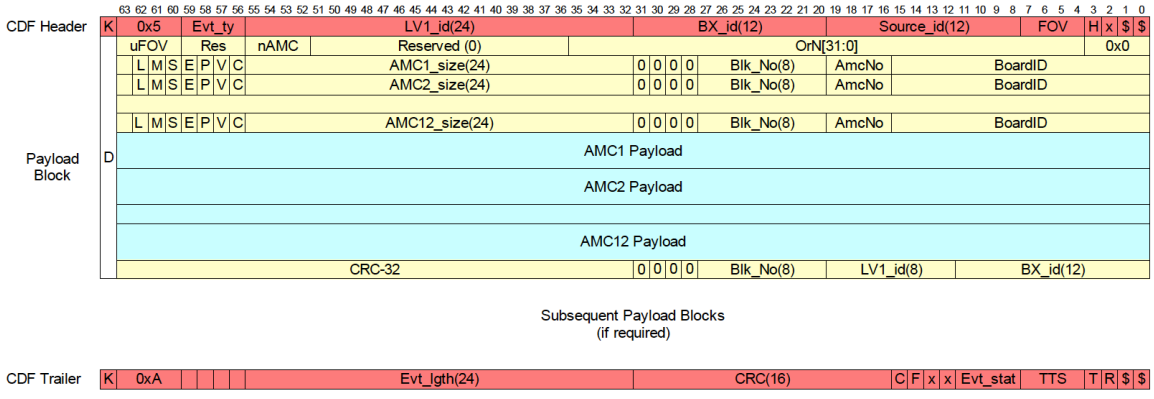


Figure C.1: *AMC13 to DAQ data format.*

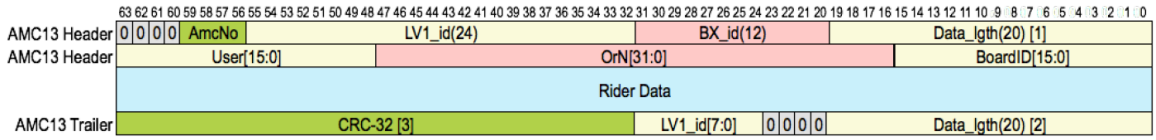


Figure C.2: *Rider to AMC13 data format.*

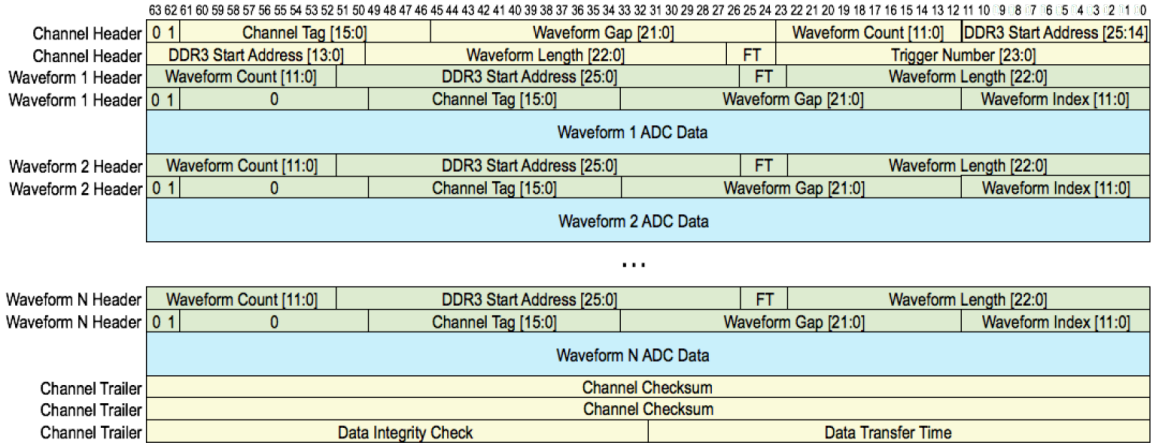


Figure C.3: *Rider Channel data format.*

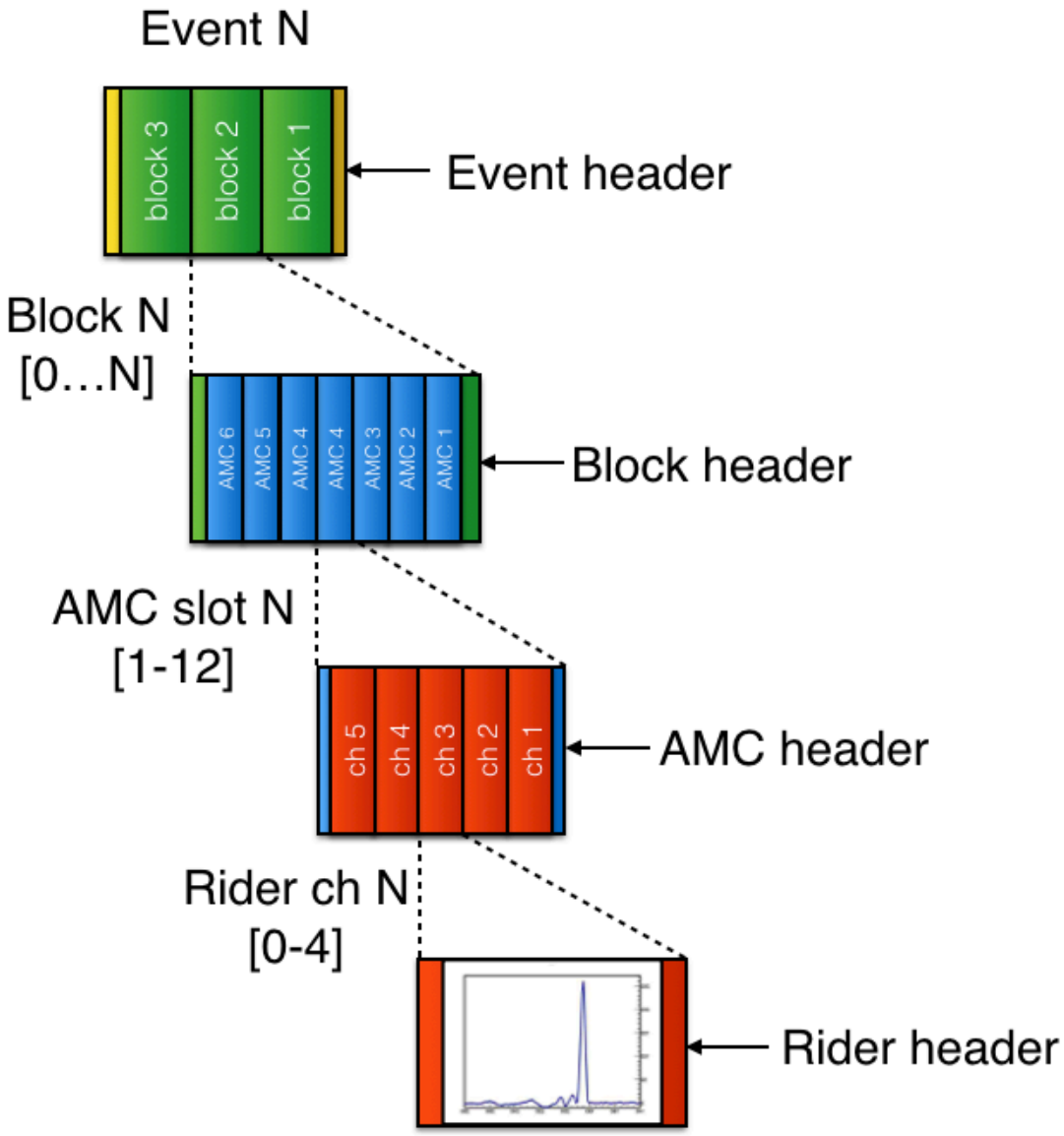


Figure C.4: *Per event data format*

Bibliography

[1] A. Fienberg et. al., *Studies of an array of PbF_2 Cherenkvo crystals with large-area SiPM readout*, Nuclear Instruments and Methods in Physics Research Section A, Vol. 783, 12-21 (2015)

[2] Rene Brun and Fons Rademakers, *ROOT - An Object Oriented Data Analysis Framework*, Proceedings ALHENP'96 Workshop, Lausanne, Sep. 1996, Nucl. Inst. & Meth. in Phys. Res. A 389 (1997) 81-86. See also http://root.cern.ch.

[3] K. S. Khaw, *Offline structure group report*, FNAL E989 g-2 Experiment Document, FM2-doc-3781-v3 (2016). http://gm2-docdb-fnal.gov:8080/cgi-bin/ShowDocument?docid=3781

[4] TRIUMF MIDAS homepage, accessed May 13, 2016. https://midas.triumf.ca/MidasWiki/index.php/Main_Page