GLAST LAT Technical Document	Document # **LAT-TD-05595-02**	Date 16 August 2006
	Author(s) Mark Strickman, Zachary Fewtrell	
	Subsystem/Office Calorimeter	
Document Title **calibGenCAL v4 Description**		

Gamma Ray Large Area Space Telescope (GLAST)
Large Area Telescope (LAT)
Calorimeter Calibration Software
calibGenCAL v4
Description

Change History Log

Revision	Effective Date	Description of Changes
2	8/16/2006	Updated to cover calibGenCAL v4 series

Contents

1. Purpose

This document describes the LAT calorimeter calibration process in the SAS environment, as applied during the Integration and Test phase of the mission, using version v4 of the calibGenCAL package.

2. Scope

This document covers calibration procedures to be used on the flight calorimeters prior to launch. In particular, it covers calibrations performed by the calibGenCAL v4 package.

This document includes discussions of both charge injection and muon calibration processes, but does not discuss on-orbit calibrations using galactic cosmic rays. It will discuss the calibration algorithms, inputs, and products . It deals primarily with software in the Science Analysis Software (SAS) environment, although topics from on-line analysis will be touched on as required.

3. Definitions

3.1. Acronyms

CAL	Calorimeter
CDE	Crystal Detector Element
CSV	Comma Separated Values
DAC	Digital to Analog Converter
CIDAC	Onboard Charge Injection DAC
EM	Engineering Model
FLE	Cal (Fast) Low Energy Discriminator
FHE	Cal (Fast) High Energy Discriminator
LAC	Log Accept Discriminator
LAT	Large Area Telescope
GLAST	Gamma-ray Large Area Space Telescope
PDA	Pin Diode Assembly
SAS	Science Analysis Software
TKR	Tracker
ULD	Upper Level (Range) Discriminator

3.2. Definitions

mm millimeter

Simulation To examine through model analysis or modeling techniques to verify conformance to specified requirements

4. References

LAT-TD-00035-1 “LAT Coordinate System.”

LAT-MD-4187 “Electronic and Muon Calibration Definition”

5. Measurement and Labeling Conventions

5.1. Coordinate system

Dimensions and values shall use the LAT Coordinate System as their reference for describing orientations and directions (if applicable). This is detailed in LAT-TD-00035-1 “LAT Coordinate System.”

5.2. Alternating layers

Calorimeter crystals are arranged in alternating layers. In the top layer, the long axis of each crystal is parallel to the X-axis, in the next, the long axis of each crystal is parallel to the Y-axis, on so on. The former are referred to as X layers and the latter as Y layers. Dimensions referred to as “transverse” without a specification of X or Y dimension are typically the same for crystals in X and Y layers. In X layers, a transverse length would be parallel to the Y-axis, while in Y layers, it would be parallel to the X-axis. Frequently, the variable name will specify X or Y, but the dimension can apply to both.

5.3. Units

The dimensions listed here are nominal and in millimeters and do not reflect tolerances. When a number is underlined in a drawing (e.g. .360.5) it means that its value has been changed by hand for the purposes of this document.

5.4. Diode labeling

Diode labeling has not been absolutely consistent throughout the development process. As a result synonyms exist for both diode size and crystal end notation. In particular, the low energy diode corresponding to the LEX8 and LEX1 channels can be referred to as “low energy”, “large” or “big”. These terms are used interchangeably. The high energy diode, corresponding to the HEX8 and HEX1 channels, can be referred to as “high energy” or “small” interchangeably.

The crystal end in the negative coordinate (X or Y) direction is referred to as “negative”, “N”, “minus” or “M” interchangeably. The crystal end in the positive coordinate direction is referred to as “positive”, “plus” or “P” interchangeably.

6. Description of the Calorimeter Calibration Process

The calorimeter calibrations are intended to allow the calculation of various calorimeter responses which, in turn, allow conversion of measurements in "instrument units" such as ADC units into "physical units" such as MeV. This is accomplished by performing measurements that produce known input and measuring the response.

For the calorimeter, the primary quantities that require calibration are:

1. The energy scale (what deposited energy corresponds to a given output in ADC units)
2. The event position scale (where along a crystal was the energy deposited) and
3. The trigger threshold performance (position and efficiency of the trigger threshold)
4. calibGenCAL is also used to generate Cal threshold CIDAC settings based on online calibration data.
5. Calibrations taken with the Cal in muon gain configuration will be extrapolated to flight gain settings.

The calibrations that are required flow from the process by which we reconstruct these quantities from data for a single CDE, and also the inverse process which involves simulating digital output from Monte Carlo simulation based energy deposit data.

6.1. CIDAC Energy Scale

Note that many of the processing steps and calibrations described use the "CIDAC" energy scale. This scale, named after the *Charge Injection DAC* settings that control the size of a calibration pulse. CIDAC is a linear charge scale, similar to the "electrons" scale used previously, although with different units. CIDAC units are proportional to the number of electrons (i.e. charge) deposited at the input of the front end processor. There are actually two distinct CIDAC scales per crystal face, one for the large (LEX) diode and one for the small (HEX) diode. Within each diode, the software that converts to CIDAC units (e.g. adc2cidac) automatically scales for range, resulting in a single CIDAC scale (e.g. for LEX8 and LEX1). Using the CIDAC scale removes dependence on the front end electronics calibration.

Note: In the past CIDAC was often referred to as simply DAC with regards to Cal offline software. We now use the term CIDAC to better differentiate between the onboard charge-injection

CALORIMETER CALIBRATION DATA FLOW

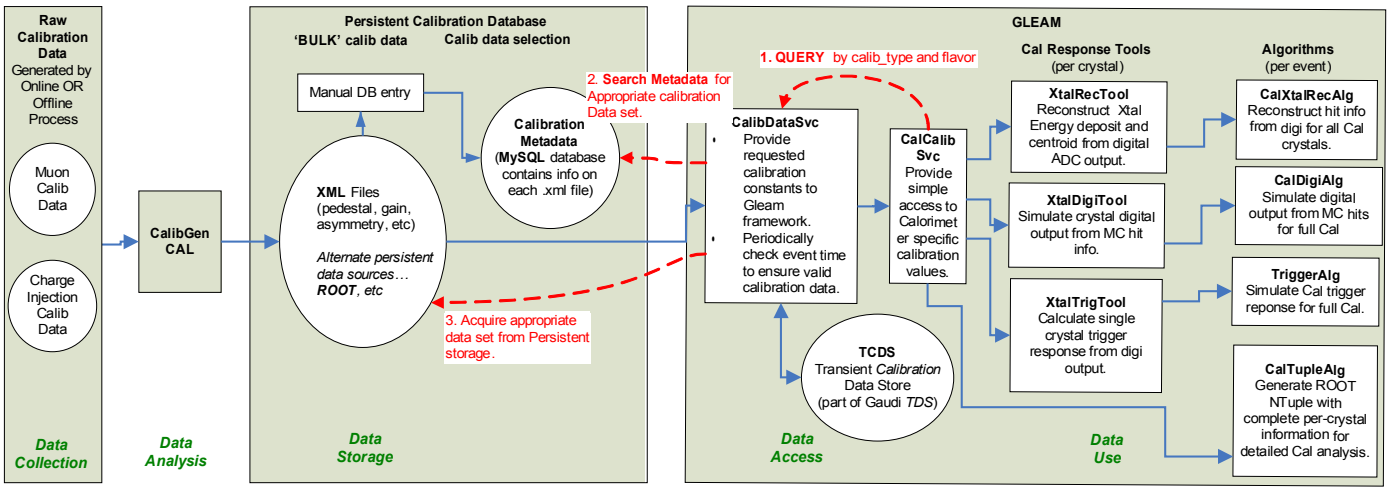


Figure 1 Overall flow of CAL calibration data

6.2. Flow of Calibration Data

Figure 1 summarizes the overall flow of calibration data in the SAS environment. This flow can be broken down into five basic steps:

1. Data Collection: Data are collected via a variety of on-line test procedures described in LAT-MD-04187. These procedures produce a variety of outputs, including html-formatted reports, CSV lists, LDF files and (after an off-line conversion) Root Files.
2. Data Analysis: Data analysis tasks are split between on-line and SAS environments. Analyses performed online normally require information not available in the off-line environment (e.g. charge injection DAC settings in the integral nonlinearity charge injection tests). Most analyses, however, are performed in the SAS environment, using the CalibGenCAL package, described in detail below. Even analyses performed on-line are "passed through" CalibGenCAL in order to convert them to standard output formats. CalibGenCAL outputs results in XML files recognized by the overall SAS calibration system.
3. Data Storage: LAT calibration data persistent storage is currently in the form of XML files, each of which contains a single calibration type for all channels and towers present during the time interval for which it is valid. calibGenCAL writes these XML files directly. The data access system is designed to read only a single XML file for each data type. At some future point, the XML files currently used may be replaced with Root files.
4. Data Access: As indicated in Figure 1, access to the calibration data by the CAL response tools is via a hierarchy of interfaces. CalCalibSvc is the API that the tools access directly. Information is passes that uniquely identifies the type, channels and time period of the required calibration data. In addition, multiple versions are supported via the "flavor" parameter.

CalibDataSvc is the GLEAM service that, given information supplied by CalCalibSvc, searches the MYSQL metadata database to find the appropriate persistent files, reads those files, and moves the data to the Gaudi Transient Calibration Data Store (TCDS), from which they can be accessed by Gaudi applications such as CAL Response Tools.

Further information on the Gaudi calibration system can be found at:

http://www-glast.slac.stanford.edu/software/calib

5. Data Use: Section below describes the processes by which energy and position are determined for each hit crystal. These processes are implemented by the CAL Response Tools.

7. calibGenCAL v4 Overview

7.1. Muon Calibrations

calibGenCAL v4 performs a variety of muon calibrations. These include:

1. *Pedestals* – Noise pedestal position and width are measured.
2. *Light Asymmetry* – Light asymmetry vs position represented by a table of 10 points, each position bin representing 1/12 of the crystal length (the end bins are not saved due to large systematic uncertainties). Asymmetry is saved for all possible combinations of big and small diode for each CDE.
3. *MevPerDac* – MevPerDac is the ratio of energy deposited by a vertical incidence muon to the muon Landau peak most probable value in sqrt($CIDAC_{minus}$*$CIDAC_{plus}$) units. It is, in effect, the bin width of the CIDAC scale in MeV and, as such, is proportional to the inverse of the gain. Because the geometric mean of the signals from each end (i.e. sqrt($CIDAC_{minus}$*$CIDAC_{plus}$)) estimates energy deposition very close to independent of hit position along the crystal, MevPerDac is not a function of position.

7.2. Charge Injection Calibrations

1. The *INTNONLIN* calibration analysis returns a table of ADC vs CIDAC values (or the inverse). Hence, it describes the relationship between the nonlinear ADC pulse height scale that comes out of the instrument and the linear CIDAC scale described in Section .
2. *TholdCI* calibration analysis calculates LAC, FLE, FHE, and ULD discriminator threshold levels in ADC units for a particular instrument configuration. These calibrations are primarily based on charge injection calibrations from the calibDAC online test suite.

7.3. DAC settings

calibGenCAL v4 is responsible for generating onboard LAC, FLE, FHE, and ULD discriminator DAC settings based on calibrations from the calibDAC online test suite.

8. calibGenCAL release plans

The calorimeter muon and charge injection calibration processes have been gathered together in the calibGenCAL package. Although this package is a part of EngineeringModel, and BeamTestRelease releases, it is independent of the Gaudi framework and GLEAM. The current version of calibGenCAL, as described in this document, is v4r3.

v3 of calibGenCAL is documented in **LAT-TD-05595-01** . v4 of calibGenCAL includes a significant body of python code which is responsible for generating offline threshold calibrations, online discriminator DAC settings, calibration validation, as well as miscellaneous data analysis and conversion tasks. The C++ code in v4 of calibGenCAL is still responsible for computationally intensive event processing. The calibration algorithms, data outputs have not changed significantly from v3. The most significant improvement is the capability of analyzing multiple Cal modules in parallel. Minor improvements include the following: more modular code structure, simpler configuration files, simpler overall calibration procedure, and significantly reduced processing time as a result of parallel analysis.

9. calibGenCAL v4 Algorithms

9.1. Phase I -- Pedestals

The pedestal analysis flow is shown in Figure 2 Pedestal Data Flow. It consists of two iterations. In the first (Phase Ia), pedestals histograms are filled using LEX8 channels from all large diodes, whether the crystal involved was hit or not. The resulting distributions may be slightly distorted by hit crystal signals, but are still representative of the pedestal distributions. The resulting histograms (one per large diode) are fit with a Gaussian model and the results output to a temporary text file.

The second phase (Phase Ib) uses the rough pedestals from Phase Ia to select crystals with signals within 5 (rough pedestal distribution) sigmas of the pedestal position. The analysis is then repeated as above, but for all channels, resulting in 4 pedestal values for each crystal end (two for each diode). These are then written out to text and XML files.

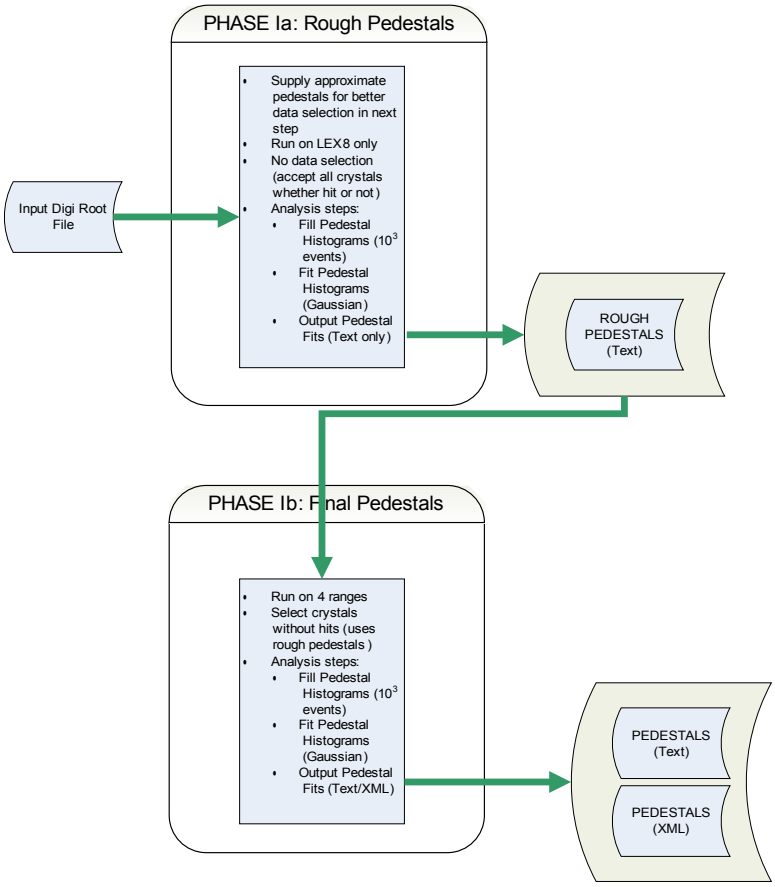


Figure 2 Pedestal Data Flow

9.1.1.Input

genMuonPed.exe requires muon collection files with at least *some* non-zero suppressed, 4 range readout Cal data. Currently, most LAT muon collections contain a periodic trigger which interleaves these types of events with the data stream which is often configured differently. The Cal should be configured in muon gain. *genMuonPed.exe* needs 10^3 usable events, so the statistics requirement is not high. Nonetheless, multiple input filenames may be space delimited in the config file.

9.1.2. Output

Pedestal results, including peak positions and widths, are stored in both a text file and an XML file. The pedestal distribution peak and width (sigma) are in adc units. They are supplied for each range of each crystal end. Format of the CAL_Ped XML file is defined in http://www-glast.stanford.edu/cgi-bin/viewcvs/calibUtil/xml/calCalib_v2r3.dtd?view=markup&rev=1.1**:**

9.1.3.Phase II -- Asymmetry

The asymmetry analysis flow is shown in Figure 3 Asymmetry Data Flow. Preliminary data selection is performed in the twrHodoscope class as described in Section . In addition, hits where the CIDAC energy scale value ≤ 0 for either face of a hit crystal are rejected. For asymmetry analysis, events that have a "goodXTrack" are used to measure asymmetry in the Y layers and vice versa i.e.

X is the "test direction" for the measurement of asymmetry in Y-layer crystals. This nomenclature makes more sense if we consider that the asymmetry in a Y-layer crystal is a function of X position. Particles that traverse a single column in X, for example, all cross Y-layer crystals at more or less the same longitudinal position and with little or no longitudinal component to their track, even though the Y crystals sampled by such an event may or may not be in a single column.

Asymmetry measurements are only performed for the middle 10 crystal-width-bins along each crystal. In other words, any event with a track in the first or last column of the "test direction" is rejected. Asymmetry at the ends of the crystal is susceptible to multiple systematics and is not calibrated. Since crystal end hits can be detected by other means than asymmetry (e.g. direct diode deposition), and, knowing that the hit was near the end of the crystal, assumptions made about hit position, asymmetry calibration near the ends of crystal is not required.

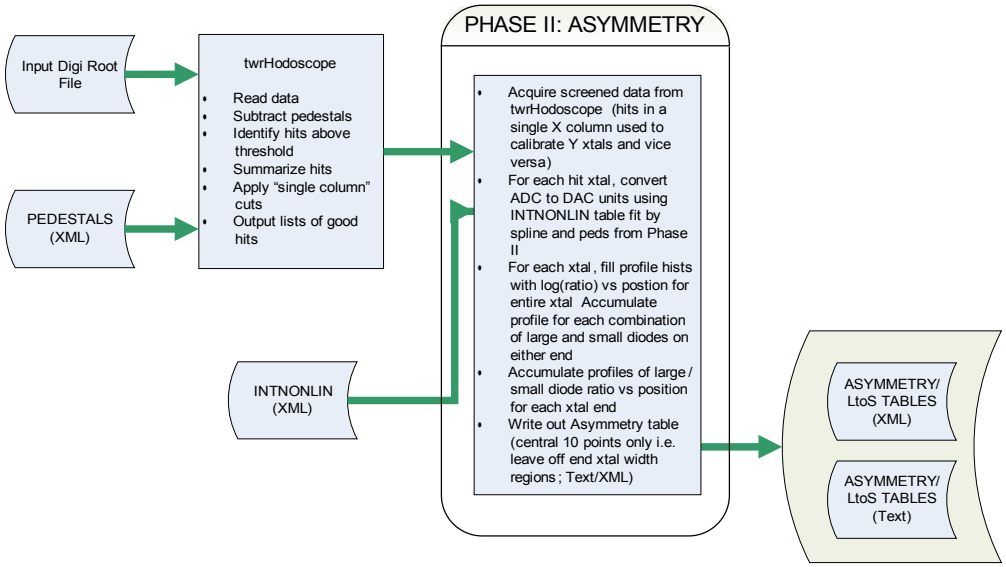


Figure 3 Asymmetry Data Flow

9.1.4.Data Selection

The twrHodoscope class performs access, logging and preliminary screening of hits for each event. It is used for both asymmetry and MevPerDac analysis. For each hit it extracts crystal ID information and ADC values, then pedestal subtracts and performs a series of tests. Note the following definitions used in selection:

- Test direction – Layer set (either X or Y) for which the one and only one hit crystal per layer align in a single column
- Longitudinal direction – The layer set (either X or Y) that is not the test direction i.e. Y if the test direction is X and vice versa

For each hit crystal, twrHodoscope performs the following tests:

- Compares the sum of positive and negative ends to a threshold to determine if the hit is valid

- Rejects any event with more than 2 hit crystals in any layer (to prevent events that escape the sides of crystals)
- Defines good[XY]Track to be true if:
    - Number of hit [XY] layers (i.e. test direction layers) = 4
    - Number of hit [XY] columns (i.e. test direction columns) = 1
    - Number of hit [YX] layers (i.e. longitudinal direction layers) > 0

  Where goodXTrack defines X to be the test direction and Y the longitudinal direction and goodYTrack defines Y to be the test direction and X the longitudinal direction.

9.1.5.Input

genMuonAsym.exe requires muon collection files with 4 range readout Cal data. The Cal should be configured in muon gain. *genMuonAsym.exe* needs very high statistics, multiple input filenames may be space delimited in the config file. *genMuonAsym.exe* ideally will need 10^4 post-cut muon hits per crystal. The 16 tower lat with 3e7 events will generally generate around half this much data. To get best available statistics, all usable input files should be included in the config file.

9.1.6.Output

The asymmetry product is just the histogram of asymmetry (log(P/M)) vs position for LE and HE diodes for each crystal. Each table contains ten numbers representing ten positions defined by the centers of the orthogonal crystals, excluding the end crystals. Currently, the read routine for the table linearly extrapolates from the last two positions to get values at the ends. Format of the CAL_Asym XML file is defined in http://www-glast.stanford.edu/cgi-bin/viewcvs/calibUtil/xml/calCalib_v2r3.dtd?view=markup&rev=1.1 **:**

9.1.7.Phase III – MevPerDac

The MevPerDac analysis flow is shown in Figure 4 MevPerDAC Data Flow. MevPerDac defines the bin width in MeV of the linear CIDAC unit energy scale described in Section . As for asymmetry, the twrHodoscope class is used to perform preliminary data selection. However, in this case, events with a "goodXTrack" are used to calibrate X crystals and those with a "goodYTrack" are used to calibrate Y crystals i.e. X is the "test direction" for X crystals and Y is the "test direction" for Y crystals. This selection restricts path length corrections to a single dimension. In addition, the "longitudinal direction" layers must have at least two layers hit to obtain a good path length correction.

Following selection, a rough track is fit to each event, using the X and Y crystal positions (i.e. using the hodoscopic nature of the calorimeter). Events where this track is more than 52.5 degrees from the vertical are rejected. Likewise, events within a crystal width of the end of any crystals are rejected.

For events not rejected above, using INTNONLIN results from genCIDAC2ADC, ADC signals from each diode are converted to the CIDAC energy scale (recall that there is one CIDAC scale for each diode). The CIDAC values from either end are then used to form an asymmetry, which in turn can be used to determine a hit position along the crystal. These positions are used to refit the particle

track and get a better track estimate. Using the better track fit, energy deposits are corrected for path length through each crystal.

For each crystal, histograms are collected of deposited energy measured with the large diode. The energy is estimated by the geometric mean of the CIDAC values at either end of the crystal:

meanCIDAClarge = sqrt(CIDAC large pos * CIDAC large neg)

Fitting the resulting peak with a Landau distribution yields a most probable value parameter, which in turn gives:

MevPerDac(large diode) = 11.2 MeV/MPV of Landau

Since muon signals are rather small in the small diode, Landau fits to individual energy deposit histograms are vulnerable to systematic and statistical errors. Instead, we form a profile histogram of meanCIDACsmall vs meanCIDAClarge using all crystals and selected events. The slope of this histogram is the ratio of MevPerDac(small diode)/MevPerDac(large diode) = large2small, which gives, for each crystal:

MevPerDac(small) = MevPerDac(large) * large2small

The resulting MevPerDac(large/small) for each crystal are written out to text and XML files.

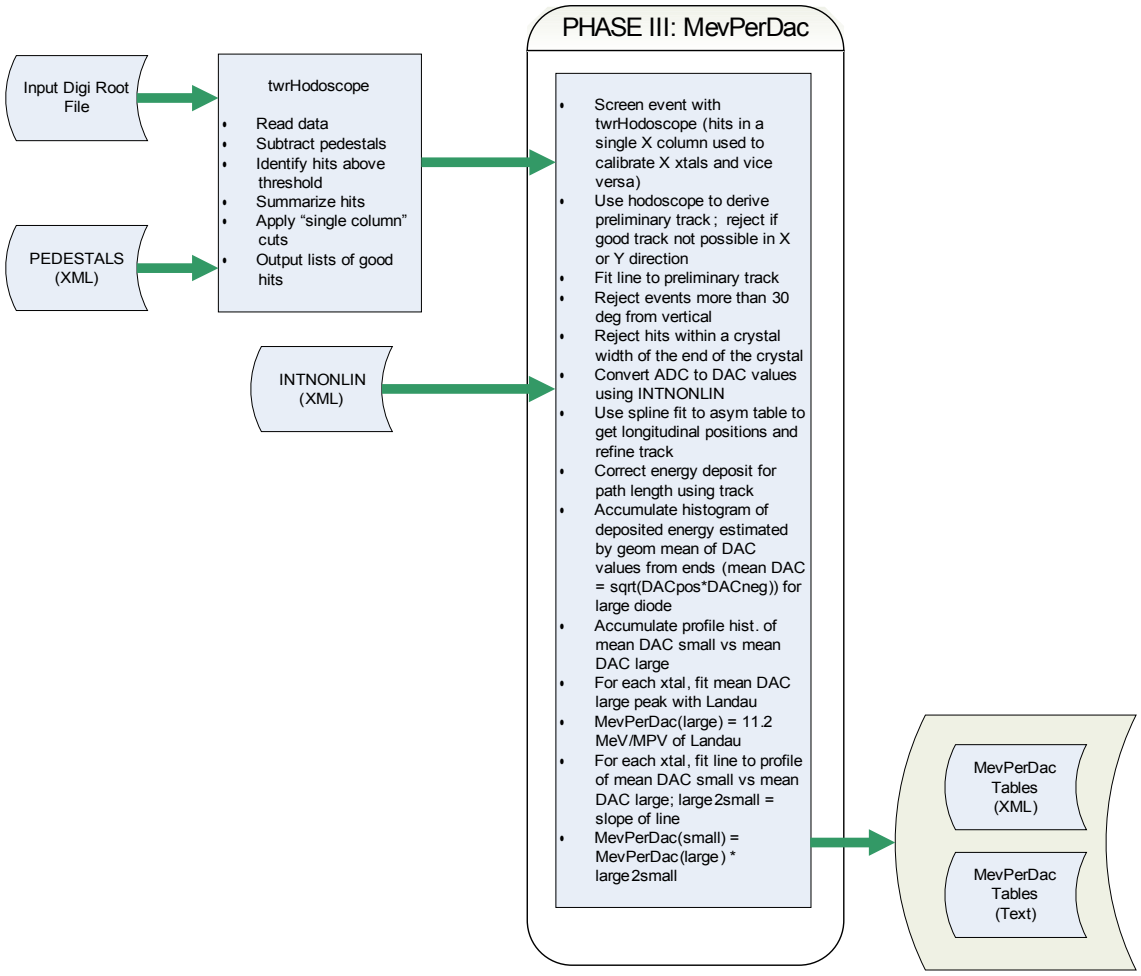


Figure 4 MevPerDAC Data Flow

9.1.8.Input

genMuonMPD.exe requires muon collection files with 4 range readout Cal data. The Cal should be configured in muon gain. *genMuonMPD.exe* needs 3e3 post-cut muon hits per crystal. The 16 tower lat with will generally generate this much data after 1.5e7 to 2e7 events. To get best available statistics, all usable input files should be included in the config file.

9.1.9.Output

The MevPerDac product is a table of MevPerDac values for each diode size for each crystal (total of two per crystal). It is stored in XML and text files. Format of the CAL_MevPerDAC XML file is defined in http://www-glast.stanford.edu/cgi-bin/viewcvs/calibUtil/xml/calCalib_v2r3.dtd?view=markup&rev=1.1 :

9.2. Charge Injection INTNONLIN Calibration (genCIDAC2ADC)

The genCIDAC2ADC routine analyses data from charge injection tests designed to measure the integral linearity of the pulse height scale. Deviations from linearity, or INTNONLIN, are used as

part of the energy scale calibration process. The INTNONLIN results are used in both the asymmetry and MevPerDac calibrations described above, and are an integral part of CalRecon as well.

9.2.1.1 Input digiRoot Files

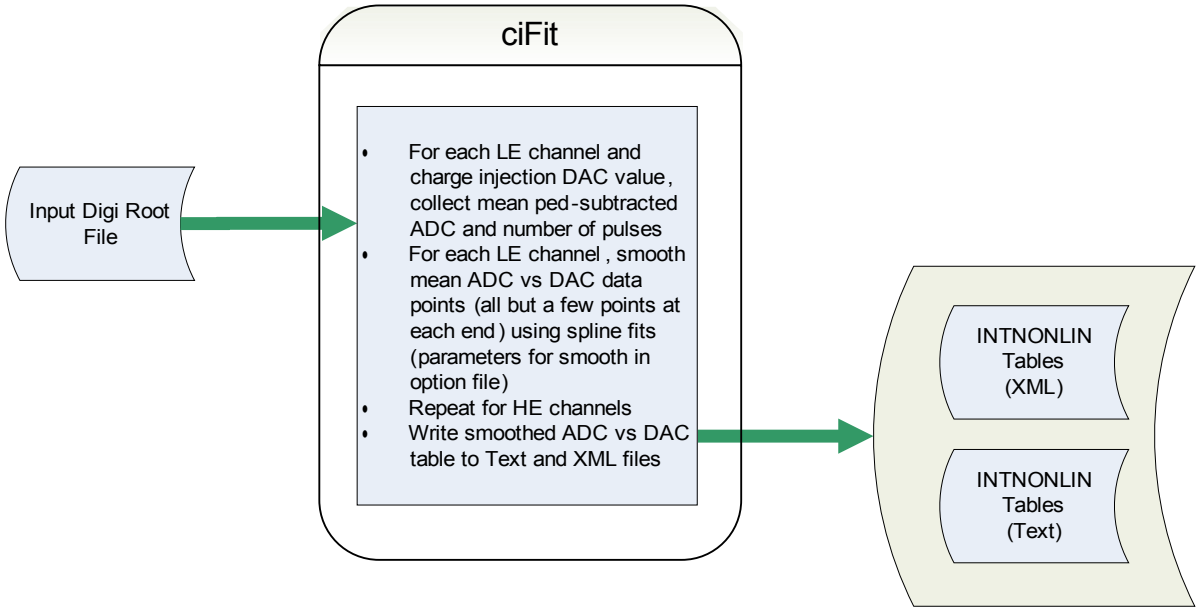
genCIDAC2ADC uses the results of the calibGen suite of tests run in the LATTE environment. See LAT-MD-04187, "CAL Electronic and Muon Calibration Suite Definition" for details. Output of these tests are LDF (or, earlier, CED) data files, which are then converted to Root using either ced2root or follow-on LDF converters.

For genCIDAC2ADC, we use element 1 of the CALF_COLLECT_CI_SINGLEX16 runs for the low energy linearity calibration and element 6 for the high energy linearity calibration. Both runs are set up with leGain = 5, heGain = 0, tackDelay = 104, fleDAC = 127 and fheDAC = 127. Element 1 is set up for leOnly, while element 6 is set up for heOnly and has Calib Gain Off.

9.2.1.2 Algorithm

genCIDAC2ADC is a relatively simple application that computes the mean ADC value for each channel in a CAL module for each charge injection DAC value, then smooths the resulting table of ADC vs CIDAC using spline functions and outputs the smoothed table of values. The smoothing process removes small, nonsignificant fluctuations from the data.

genCIDAC2ADC currently does not make corrections due to the difference in cross-talk-induced nonlinearities between charge injection and muon calibrations. This is because the bulk of data currently collected are collected with FLE set high (i.e. externally triggered data), for which this phenomenon is well clear of the muon peak in the spectrum. This correction will be included in a future version of the code.



9.2.2.Products

genCIDAC2ADC produces both text and XML calibration files containing both the CIDAC values used in the test and the smoothed mean ADC values corresponding to those CIDAC values. These tables of values can then be used by functions such as adc2cidac to convert ADC results into the linearized "CIDAC" pulse-height scale. Format of the CAL_IntNonlin XML file is defined in http://www-glast.stanford.edu/cgi-bin/viewcvs/calibUtil/xml/calCalib_v2r3.dtd?view=markup&rev=1.1 **:**

9.3. TholdCI

Cal offline threshold calibration is based on the characterization output of the calibDac online test suite. *LAC, FLE, FHE,* and *ULD* thresholds are calibrated in ADC units. The process is essentially a lookup, from a given instrument configuration file (provided in xml snapshot form) into the appropriate xxx2adc.xml online threshold characterization file from the calibDac test. For time saving reasons, calibDac currently populates the characterization curves sparsely: calibGenCAL provides interpolation and smoothing routines to facilitate simple lookup by *tholdCIGen*.

9.4. DAC Settings

Cal online DAC settings are mainly generated from output of the calibDac online test suite. *LAC, FLE, FHE,* and *ULD* thresholds are set in discriminator DAC units. Input files are smoothed xxx2adc characterization curves, pedestal, and relgain calibrations from calibDac suite as well as adc2nrg and FleFheBias output from muon calibration in calibGenCAL. adc2nrg is overall ratio of energy deposited in CID to digital adc output per channel. FleFheBias calibrates bias between FLE and FHE trigger thresholds measured with charge injection verses those measured with muon energy deposit.

Algorithm for LAC,FLE,FHE DAC setting evaluation is as follows:

1. Take requested threshold in MeV units and convert to ADC scale using adc2nrg xml file.
2. Apply any trigger bias to this value using FleFheBias this adc value
3. Look up appropriate DAC setting in xxx2adc characterization curve

Algorithm for ULD setting evaluation is somewhat different. A single ULD DAC per crystal face affects all 4 ADC ranges.

1. For given crystal face, find ADC range with lowest ADC saturation point in DAC scale from uld2adc characterization file.
2. Subtract requested adc_margin from this saturation point (default = 50 adc units)
3. Select new DAC value by looking up new adc limit in appropriate uld2adc curve.

10. calibGenCAL v4 Software Environment

10.1.C++ based executables

Required software configuration

calibGenCAL C++ executables are designed to run with the GLAST offline software environment. This software environment uses the CMT tool for package management and the software is distributed in 'Release Packages' which specify a consistent set of sub-packages and their versions which have been tested to work together. Currently calibGenCAL ships as part of the EngineeringModel and BeamTestRelease packages. calibGenCAL requires a fully functioning CMT environment and appropriate version of a GLAST SAS release package both for execution as well as development.

GLAST makes use of several front ends for the CMT package management system, including VCMT, MRvcmt, and glastpack. This scope of this document does not include operation of the CMT package management system or these front end tools. Information specific to calibGenCAL's implementation of this system will be given, but a general understanding of the system is assumed.

Along with GLAST SAS system: calibGenCAL supports both Windows and Linux environments.

To run executables

calibGenCAL specifies in it's CMT requirements file that it's executables should be appended to the system path. The *'cmt config'* command, or its GUI equivalent, will generate setup scripts which will perform this task. Incidentally, the 'launch shell' command in MRvcmt will run a system shell with the appropriate environment variables set for execution of all calibGenCAL programs.

With the environment properly configured, the user may invoke calibGenCAL executables from any directory simply by typing the executable name:

Example: `"> genMuonPed.exe calibGenCAL_suite.cfg"`

10.2.Python scripts

Required software environment

calibGenCAL python scripts are developed with Python version 2.4 on both Linux and Windows. They make use of 2 3rd party python libraries which are also used elsewhere in GLAST software. If these libraries are not available on your system, then windows installers are readily available online as are Linux versions. Most modern Linux distributions will offer these packages optionally through their default software installation system. These libraries are:

- Numeric
- 4Suite-XML
- PyROOT (generally available with the ROOT distribution included with all SAS software)

To run scripts

calibGenCAL python folder is placed on the system PATH along with the binary executable folder. Every calibGenCAL python script is supplied with a Windows .bat file launcher script which sets up the environment and launches the appropriate script as well as a Linux .sh script which performs the same task. The launcher scripts pass all command line parameters onto associated python scripts. All that needs to be done to run a calibGenCAL python script from the command line in any directory is to invoke the associated launcher script. Here are examples for the pedVal.py python script:

Windows: "pedVal.bat peds.xml"

Linux: "pedVal.sh peds.xml"

Note: All calibGenCAL python scripts will print their command line usage specifications to screen if they are invoked with no arguments. Here is an example from Linux command line using pedVal.py script:

```
[fewtrell@ossep9 ~/tmp]$ source calibGenCAL/v4r3/cmt/setup.csh
[fewtrell@ossep9 ~/tmp]$ pedVal.sh
ERROR:pedVal:pedVal [-V] [-r] [-L <log_file>] [-R <root_file>] <xml_file>
```

11. End 2 End Calibration Procedure Run Through

This document covers the steps required to generate a full set of calibrations and Cal DAC configurations using calibGenCAL software. It will enumerate the application, inputs, and outputs for each step. It will not necessarily go into complete detail for running each application, but it should allow someone who is familiar with GLAST and GLAST offline software to complete the process with reasonable extra effort.

11.1.CIDAC2ADC processing from calibGen output

Obtain input files

First, identify proper calibGen run and select proper digi event files. calibGen runs singlex16 calibration tests in several configurations. For this stage, we need the two files w/ the following configuration:

```
legain=5 hegain=0 le_only calibGain=on fle=nominal
legain=5 hegain=0 he_only calibGain=off fle=nominal
```

As of the writing of this document, the calibGen run id's for these tests are 201 and 204 respectively. Other calibGen suite versions may use different run order or instrument configurations, user must confirm the correct configurations.

Generate Configuration File

Fully populated example configuration file is located via WebCVS: http://www-glast.stanford.edu/cgi-bin/viewcvs/calibGenCAL/cfg/calibGenCAL_suite.cfg?view=markup&rev=1.2&pathrev=v4r3: *(Figure 5 calibGenCAL_suite.cfg)*

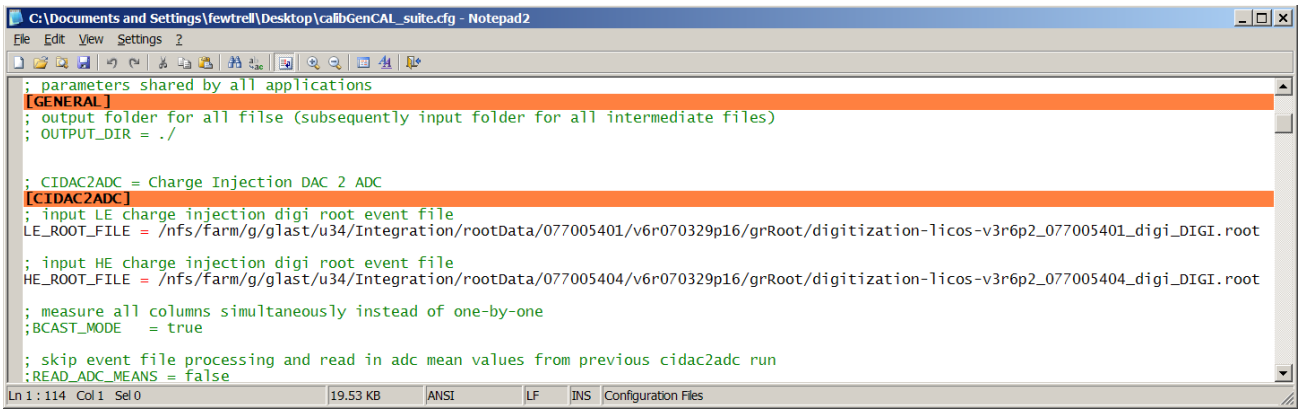


Figure 5 calibGenCAL_suite.cfg

calibGenCAL_suite.cfg is shared by all calibGenCAL C++ applications. It is .ini or .cfg format, which was chosen for ease of human editing. It contains separate sections for each application, for this stage we are interested in **[CIDAC2ADC]** and **[GENERAL]** sections. Output filenames are auto generated and dependent applications will refer to the **[CIDAC2ADC]** section in order to auto-generate input filenames.

Run Application

Assuming correct CMT configuration, command line is simple:

```
"> genCIDAC2ADC.exe calibGenCAL_suite.cfg"
```

Outputs

genCIDAC2ADC.exe will generate *cidac2adc.xxx.txt* table file as main output. This output file will be referenced by later C++ stages including genMuonAsym.exe and genMuonMPD.exe. Ancillary outputs include ROOT file with histograms for analysis.

Post Processing

- Generate official XML output file with python script **inlTXT2XML.py.** Along with being a final calibGenCAL product, this file will be referenced by later stages of python script processing within calibGenCAL including *tholdCIGen.py*.
- Validate and generate summary plots from xml with python script **intNonlinVal.py**

11.2.Muon Pedestal calibration

Obtain input files

Obtain suitable digi event files as specified in . *genMuonPed* will also require the *cidac2adc.xxx.txt* from the previous stage

Generate Configuration File

Use the same *calibGenCAL_suite.cfg* file as *genCIDAC2ADC.exe*, but this time populated the [ROUGH_PEDS] and [MUON_PEDS] sections. *genMuonPed.exe* will also reference the [CIDAC2ADC] section to determine input filename, so this must be populated as well.

Run Application

Assuming correct CMT configuration, command line is simple:

```
"> genMuonPed.exe calibGenCAL_suite.cfg"
```

Outputs

genMuonPed.exe will generate *muonPed.XXX.txt* table file as main output. This output file will be referenced by later C++ stages including genMuonAsym.exe and genMuonMPD.exe. Ancillary outputs include ROOT file with histograms for analysis.

Post Processing

- Generate official XML output file with python script **pedTXT2XML.py.**
- Validate and generate summary plots from xml with python script **pedVal.py**

11.3.Muon Asymmetry calibration

Obtain input files

Obtain appropriate digi event files as specified in . *genMuonAsym* will also require the *cidac2adc.xxx.txt* and *muonPed.xxx.txt* files from the previous stages.

Generate Configuration File

Use the same *calibGenCAL_suite.cfg* file as *genCIDAC2ADC.exe*, but this time populated the [MUON_ASYM] section. *genMuonAsym.exe* will also reference the [CIDAC2ADC] and [MUON_PEDS] sections to determine input filenames, so these must be populated as well.

Run Application

Assuming correct CMT configuration, command line is simple:

```
"> genMuonAsym.exe calibGenCAL_suite.cfg"
```

Outputs

genMuonAsym.exe will generate *muonAsym.XXX.txt* table file as main output. This output file will be referenced by later C++ stages including genMuonMPD.exe. Ancillary outputs include ROOT file with histograms for analysis.

Post Processing

- Generate official XML output file with python script **asymTXT2XML.py.**
- Validate and generate summary plots from xml with python script **asymVal.py**

11.4.Muon MevPerDAC calibration

Obtain input files

Obtain appropriate digi input files as specified in . *genMuonMPD* will also require the *cidac2adc.xxx.txt*, *muonPed.xxx.txt,* and *muonAsym.xxx.txt* files from the previous stages.

Generate Configuration File

Use the same *calibGenCAL_suite.cfg* file as *genCIDAC2ADC.exe*, but this time populated the [MUON_MPD] section. *genMuonMPD.exe* will also reference the [CIDAC2ADC], [MUON_PEDS], and [MUON_ASYM] sections to determine input filenames, so these must be populated as well.

Run Application

Assuming correct CMT configuration, command line is simple:

```
"> genMuonMPD.exe calibGenCAL_suite.cfg"
```

Outputs

muonMPD.XXX.txt table file as main output.

muonMPD.xxx.adc2nrg.txt table file, which is referenced by later stages.

Ancillary outputs include ROOT file with histograms for analysis.

Post Processing

- Generate official XML output file with python script **mpdTXT2XML.py.**
- Validate and generate summary plots from xml with python script **mpdVal.py**
- Generate *adc2nrg.xml* using the ***adc2nrgTXT2XML.py*** script. This file will be used in later DAC settings processing.
- Validate *adc2nrg* output with ***adc2nrgVal.py*** script.

11.5.Generate online DAC settings from calibDac output

A slightly out of date, but significantly more detailed documentation of this process is available here: http://www-glast.stanford.edu/cgi-bin/viewcvs/*checkout*/calibGenCAL/doc/gensettings_scripts.html?rev=1.7&pathrev=v4r3:

Note: From time to time, file name conventions may change. Many of our scripts attempt to auto detect input files and auto generate output filenames. Occasionally there will be a conflict in file naming conventions. If this is the case, the simplest method is to edit the config files to override the default choices, or alter the filenames to match the calibGenCAL software.

1. Collect input files

Identify appropriate calibDac online test. Collect the following input files:

- 1 xxx**lac2adc.xml**, xxx**fle2adc.xml**, xxx**fhe2adc.xml**, xxx**uld2adc.xml,** xxx**pedestals.xml,** xxx**relgain.xml** for each Cal Module. Ignore the *'smoothxxx.xml'* versions of these files created by online as our offline code will do a better job.
- Obtain a *FleFheBias.xml* file for appropriate Cal modules. We are temporarily unable to generate this calibration as the online test has changed & we have yet to develop a new calibration program to process it. Currently, the best choice is to use the most recent bias information which is available. (Best 16 tower LAT data is from Jan '06 as of writing time.)

- It is recommended to place all of these files in the same folder as many subsequent scripts will scan the folder for their respective input files.

2. build_adcsmooth

The *xxx2adc.xml* characterization files come from calibDac sparsely sampled. We have the ***adcsmooth.py*** script to interpolate and smooth this data. We also have the ***build_adcsmooth.py*** script to run this tool an a large group of files.

build_adcsmooth.cfg

A fully populated example of this file is available here: http://www-glast.stanford.edu/cgi-bin/viewcvs/*checkout*/calibGenCAL/cfg/build_adcsmooth.cfg?rev=1.1:

Modify it to contain the proper list of Cal Modules.

build_adcsmooth.py

Run ***build_adcsmooth.py*** script, which will generate a *.bat* and *.sh* script which in turn will generate a full set off *xxx2adc_filtered.xml* files.

Ancillary outputs include plots and validation material for *xxx2adc* curves.

Note: At this time it would be a good idea to move your unfiltered xxx2adc files into some other directory so as not to confuse later scripts which will scan the folder for xxx2adc files.

3. build_gensettings_cfg

This script serves the purpose of gathering all input filenames and configuration for the *gensettings.py* script.

build_gensettings_cfg.cfg

Fully populated example file is available here: http://www-glast.stanford.edu/cgi-bin/viewcvs/calibGenCAL/cfg/build_gensettings_cfg.cfg?view=markup&rev=1.1:

Make sure you edit the **[CFG1]** section to specify the proper target cal configuration.

build_gensettings_cfg.py

Run this script to generate *gensettings.cfg* file.

4. gensettings

gensettings.py servers the purpose of running *genXXXSettings.py* scripts for each Cal module and threshold type.

gensettings.py will run directly off the *gensettings.cfg* file created in previous step. It will output a *.sh* and *.bat* script which will launch the child scripts.

run_gensettings

Launch the newly create batch file to generate the full set of Calorimeter dac settings.

Validation and plot output will be generated as well.

Outputs

A series of LATTE input Cal snapshot fragment files, named *latest_xxx_fle.xml*, *latest_xxx_fhe.xml, latest_xxx_lac.xml, latest_xxx_uld.xml*. Filenames will indicate the requested threshold leve in MeV and Cal module serial number. A second set of files with timestamp filenames will also be generated, but online software requests only the *latest_xxx.xml* filenames.

11.6.Offline Threshold calibration (tholdCI)

build_tholdci_cfg

The tholdci.cfg file can be generated from *gensettings.cfg* , and *cidac2adc.xml* files, both of which were generated in previous phases. Make sure that you have proper configuration selected for gain setting, threshold levels, etc…

Alternatively it can be edited from the fully populated example file available here: http://www-glast.stanford.edu/cgi-bin/viewcvs/calibGenCAL/cfg/tholdCIGen.cfg?view=markup&rev=1.3 :

tholdCIGen.py

Run this script to generate offline *CAL_TholdCI* calibration xml file, find spec for this file here: http://www-glast.stanford.edu/cgi-bin/viewcvs/calibUtil/xml/calCalib_v2r3.dtd?view=markup&rev=1.1 :

tholdCIVal.py

Run this script to generate validation analysis & plots.

11.7.Extrapolation to flight gain

Pedestals

Flight gain pedestals can easily be obtained by re-running *genMuonPed.exe* against data taken with the Cal in flight gain configuration.

cidac2adc

cidac2adc calibration for flight mode is similarly obtained by rerunning *genCIDAC2ADC.exe*, except this time using the following runs:

```
legain=5 hegain=15 le_only calibGain=on fle=nominal

legain=5  hegain=15 he_only calibGain=on fle=nominal
```

As of the writing of this document, the calibGen run id's for these tests are 101 and 102 respectively.

Don't forget to convert to xml and validate.

calibGainCoeff

In order to extrapolate muon calibrations into flight gain, we need to calibrate the signal ratio generated by enabling the calibGain circuit for HE channels.

We begin this process by rerunning *genCIDAC2ADC.exe* on the following two files:

```
legain=5 hegain=15 he_only calibGain=on fle=127

legain=5 hegain=15 he_only calibGain=off fle=127
```

As of the writing of this document, the calibGen run id's for these tests are 106 and 108 respectively. Because they are both HE runs, you must run genCIDAC2ADC.exe *twice*, each time on HE data only. (to do this, leave entry for LE channel in calibGain_suite.cfg file empty).

Go ahead and convert the txt outputs to xml.

Feed both xml files into ***calibGainCoeff.py*** script to generate the *calibGainCoeff.txt* output table.

Asymmetry & MevPerDAC

asymApplyCalibGain.py and ***mpdApplyCalibGain.py*** scripts can be used to apply the *calibGainCoeff* factors to the muon calibrations. The resulting files will be configured for flight gain.

TholdCI

Simply rerun ***tholdCIGen.py*** this time make sure that the gain fields and dac settings files match the flight configuration.

11.8.Generate Onboard Filter calibration files

genFlightPed.py script will generate onboard formatted C header file with Cal pedestals from an offline xml CAL_Ped file as gereated in previous steps. It will require a serial number, which is by convention assigned to the row_id in the offline calibration metadatabase of the source pedestal file.

genFlightGain

1. Generate *CAL_MuSlope* offline calibration xml file with ***muSlopeTXT2XML.py*** script. This file is used in later calibration processing. Format of this file is defined here: http://www-glast.stanford.edu/cgi-bin/viewcvs/calibUtil/xml/calCalib_v2r3.dtd?view=markup&rev=1.1: The input for this script is the *adc2nrg.txt* file generated by *genMuonMPD.exe* in earlier phase. (this file is temporary as the official state for a muSlope file must be in flight gain).

2. Merge online relgain.xml calibration files into one file with ***calFitsMerge.py.***

3. Apply relgain factor to convert muSlope from gains le=5,he=0 to le=5, he=15. To do this, use the ***calCalibApplyRelgain.py*** script.

genFlightGain.py script will now generate another C header file for onboard use. In order to obtain a serial number, commit the source MuSlope file to database and use the unique row_id.

12. CalXtalResponse

CalXtalResponse is the library used by Gleam which is responsible for applying the calibration outputs of calibGenCAL

12.1.Single CDE Recon Summary

The following steps are executed by the XtalRecTool to compute the amount of and position of energy deposition in a single hit CDE (**bold** quantities are results of calibrations; *italicized* quantities are functions):

1. Rescale to CIDAC scale
    2. Subtract **Pedestals (CAL_Ped)** from ADC values from each end of CDE.
    3. Convert pedestal subtracted ADC values to CIDAC scale using function CIDAC = *adc2cidac*(ADC). This function makes use of the **Integral Nonlinearity (CAL_IntNonlin)** calibration results. The function uses a cubic spline functional representation of the calibration data points.

4. Position

    5. Calculate the "Asymmetry" LightAsym = *log*($CIDAC_{plus}$/$CIDAC_{minus}$) where $CIDAC_{plus/minus}$ is the signal in CIDAC units from the plus or minus end of the CDE respectively.

    6. Calculate position of hit along CDE using Position = *LightAsym2posPM*(LightAsym), where *P* and *M* assume values of large or small depending on whether the signals from the plus or minus ends of the CDE are taken from the large or small diodes, respectively. This function makes use of the **Asymmetry (CAL_LightAsym, CAL_muPbigMsmallAsym, CAL_muPsmallMbigAsym)** calibration results using a cubic spline functional representation of the calibration datapoints.

7. Energy

    a. If signals from CDE ends come from different size diodes, convert signals from large diode (in CIDAC units) to small diode equivalent (also in CIDAC units) using the ratio of two asymmetries (note LP is signal from large diode, positive end, LP is large diode, negative end, similar for SP, SN for small diode):

    Exp(AsymLL) = LP/LN

    Exp(AsymLS) = LP/SN

    LN/SN = Exp(AsymLS)/Exp(AsymLL)

    Similar for LP/SP

    b. Compute the geometric average of the CIDAC values from each end of the CDE. Depending on which diodes are available, this could be:

i. $CIDAC_{geom\ mean} = sqrt(CIDAC_{big\ plus} * CIDAC_{big\ minus})$
ii. $CIDAC_{geom\ mean} = sqrt(CIDAC_{small\ plus} * CIDAC_{small\ minus})$
iii. $CIDAC_{geom\ mean} = sqrt(CIDAC_{small\ equiv\ plus} * CIDAC_{small\ minus})$
iv. $CIDAC_{geom\ mean} = sqrt(CIDAC_{small\ plus} * CIDAC_{small\ equiv\ minus})$

c. Calculate the deposited energy using the **MevPerDac (CAL_muBigDiodeMevPerDac, CAL_muSmallDiodeMevPerDac)** for the appropriate diode (Big/Big equiv, Small respectively): Energy (MeV) = **MevPerDac** * $CIDAC_{geom\ mean}$

Note: **MevPerDac** is the inverse of the gain; it is, in effect, the width of a CIDAC scale bin in MeV.

8. In the future, position and energy will have to be computed using only one end of a CDE, but this is not yet supported in the software:

- Dead PDA or partial electronics failure
- Position reconstructed off end of CDE implying direct diode deposit
- TKR or CAL indication of event too close to CDE end to have good asymmetry due to transverse dependence of light yield near ends

a. In these cases, the position must be supplied externally. In the first case, TKR or CAL hodoscope data can provide a position, at least to within one CDE width. In the second and third cases, assumptions can be made that the event occurred near the end of the CDE.

b. Given a position estimate, PlusMinusRatio = *exp*(*pos2LightAsymPM*(position)). *pos2LightAsymPM*(position) is the inverse of LightAsym2posPM. It uses the inverse of the **Asymmetry** calibration, represented by a cubic spline function, to return $\log(CIDAC_{plus}/CIDAC_{minus})$ as a function of position. P and M are as described in above.

c. If the valid end signal is $CIDAC_{minus}$, compute
$CIDAC_{plus\ estimated} = CIDAC_{minus} * PlusMinusRatio$

d. If the valid end signal is $CIDAC_{plus}$, compute
$CIDAC_{minus\ estimated} = CIDAC_{plus} / PlusMinusRatio$

e. Compute energy as in above using the measured and estimated ends.

12.2.Single CDE Digitization Summary

This process is the inverse of the CDE Recon process described in Section . The process begins with Monte Carlo integrated hits summarizing simulated energy deposits and their positions for a particular CDE crystal and completes with simulated digital output (ADC and trigger flags).

1. For each energy deposit in the crystal, sum the apparent signal into each of the 4 diodes.

   a. Use MevPerDAC calibration to determine the mean signal level for both small diodes (one per face) and both large diodes (one per face).
   b. Use Asymmetry calibration and hit position to determine ratio of signal at positive and negative crystal faces.
   c. There can be > 1 recorded MC Integrating hit per crystal, loop through them and keep a running sum in appropriate DAC scale for each diode.
2. For each energy deposit directly in a diode, update the running diode signal sum accordingly. (Using fixed constants to convert electrons in diode to deposited energy scale).
3. Add poissonic noise based on n electrons in each diode.
4. Simulate electronic noise by applying random fluctuation based on Gaussian distribution w/ width = current crystal's ADC pedestal width.
5. Calculate LAC, FLE, FHE results based on final diode signal levels and tholdCI threshold calibrations.
6. Convert CIDAC signal levels to ADC output scale.
7. Determine range selection based on tholdCI ULD threshold calibration.