GLAST LAT TECHNICAL DOCUMENT	Document # **LAT-TD-05595-03**	Date 16 August 2006
	Author(s) Mark Strickman, Zachary Fewtrell	
	Subsystem/Office Calorimeter	
Document Title **calibGenCAL v5 Description**		

Gamma Ray Large Area Space Telescope (GLAST) Large Area Telescope (LAT) Calorimeter Calibration Software calibGenCAL v5r0p1 Description

Change History Log

Revision	**Effective Date**	**Description of Changes**
2	8/16/2006	Updated to cover calibGenCAL v4 series
3	10/03/2006	Updated to cover calibGenCAL v4r4
4	04/18/2007	Updated to cover calibGenCAL v5 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 v5 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 v5 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 DAC 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 calibration DAC and the DAC settings used to specify threshold levels.

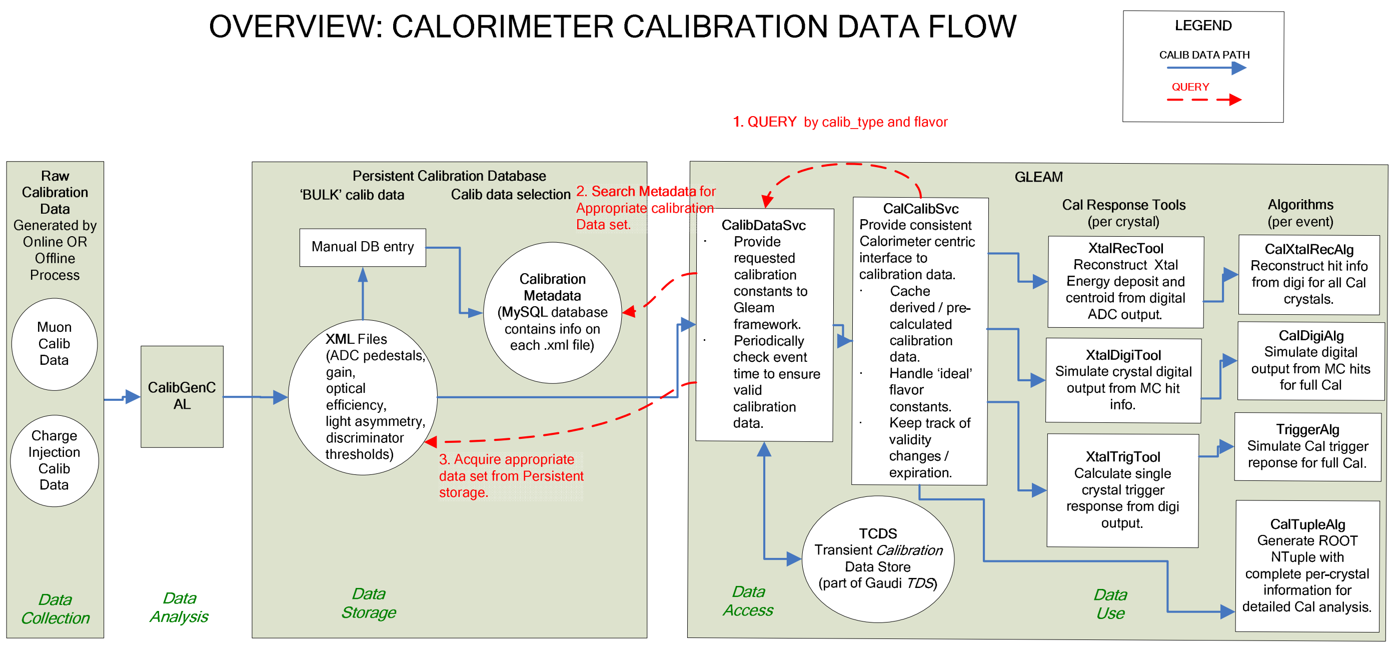

OVERVIEW: CALORIMETER CALIBRATION DATA FLOW
LEGEND
CALIB DATA PATH
QUERY
1. QUERY by calib_type and flavor
Raw Calibration Data Generated by Online OR Offline Process
Muon Calib Data
Charge Injection Calib Data
Data Collection
CalibGenCAL
Data Analysis
Persistent Calibration Database
'BULK' calib data
Calib data selection
2. Search Metadata for Appropriate calibration Data set.
Manual DB entry
Calibration Metadata (MySQL database contains info on each .xml file)
XML Files (ADC pedestals, gain, optical efficiency, light asymmetry, discriminator thresholds)
3. Acquire appropriate data set from Persistent storage.
Data Storage
GLEAM
CalibDataSvc · Provide requested calibration constants to Gleam framework. · Periodically check event time to ensure valid calibration data.
CalCalibSvc Provide consistent Calorimeter centric interface to calibration data. · Cache derived / pre-calculated calibration data. · Handle 'ideal' flavor constants. · Keep track of validity changes / expiration.
TCDS Transient Calibration Data Store (part of Gaudi TDS)
Data Access
Cal Response Tools (per crystal)
XtalRecTool Reconstruct Xtal Energy deposit and centroid from digital ADC output.
XtalDigiTool Simulate crystal digital output from MC hit info.
XtalTrigTool Calculate single crystal trigger response from digi output.
Algorithms (per event)
CalXtalRecAlg Reconstruct hit info from digi for all Cal crystals.
CalDigiAlg Simulate digital output from MC hits for full Cal
TriggerAlg Simulate Cal trigger reponse for full Cal.
CalTupleAlg Generate ROOT NTuple with complete per-crystal information for detailed Cal analysis.
Data Use

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.
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 GAUDI 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 7 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 Overview

7.1. Muon Calibrations

calibGenCAL performs a variety of calibrations on ground muon collection data. These include:

1. *Pedestals* – Pedestal position and noise distribution width are measured.
2. *Light Asymmetry* – Light asymmetry vs longitudinal 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($\mathrm{CIDAC}_{\mathrm{minus}}$*$\mathrm{CIDAC}_{\mathrm{plus}}$) 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($\mathrm{CIDAC}_{\mathrm{minus}}$*$\mathrm{CIDAC}_{\mathrm{plus}}$)) 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 6.
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 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 history

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 v5r0p1.

v3 of calibGenCAL is documented in **LAT-TD-05595-01** .

v4

- 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.

v4r5

- Introduces *genMuonCalibTkr.exe* program. This program is intended to replace *genMuonAsym.exe* and *genMuonMPD.exe* in most cases. *genMuonCalibTkr* uses Tracker track information to get more precise hit position info and to make better cuts to avoid partial hits which do not traverse
- *development stage:* Addition of alternate intNonlin processing scripts which correct for electronic crosstalk and non-linearities. These scripts are still under development

v5

- Introduction of makefile driven calibration suite. Algorithms are mostly unchanged from v4r5.
- Reason for major version # increment is that the command line interface has been modified to better facilitate Makefile implementation. calibGenCAL applications & scripts now retrieve as much configuration information as possible from command line in lieu of the config file model used previously.

9. calibGenCAL v4 Algorithms

9.1. Pedestal Calibration

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.

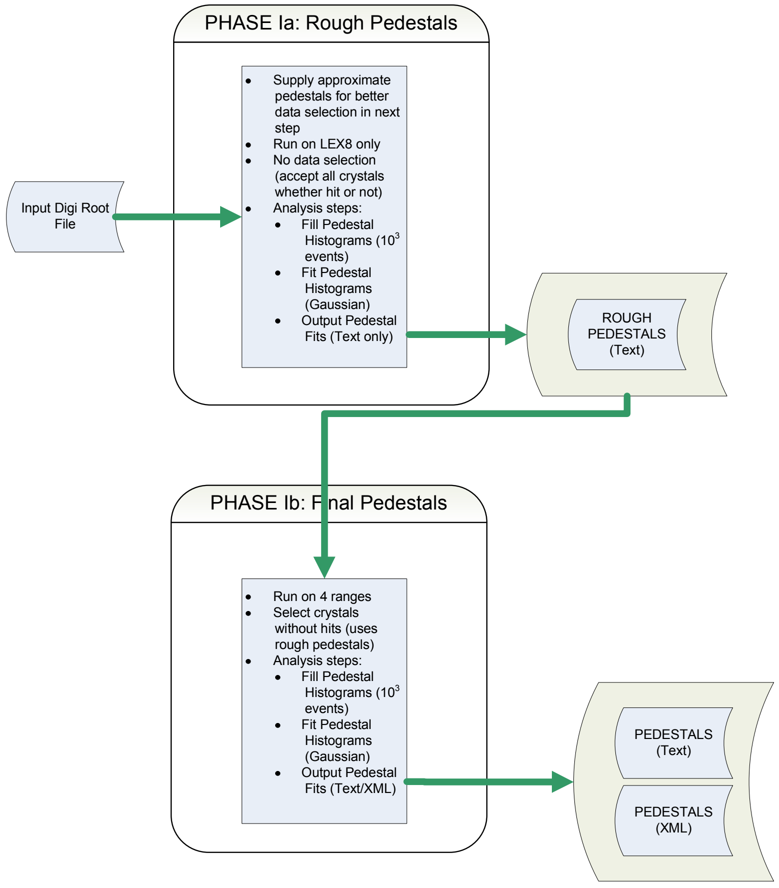

PHASE Ia: Rough Pedestals
Supply approximate pedestals for better data selection in next step
Run on LEX8 only
No data selection (accept all crystals whether hit or not)
Analysis steps:
Fill Pedestal Histograms (10³ events)
Fit Pedestal Histograms (Gaussian)
Output Pedestal Fits (Text only)
Input Digi Root File
ROUGH PEDESTALS (Text)
PHASE Ib: Final Pedestals
Run on 4 ranges
Select crystals without hits (uses rough pedestals)
Analysis steps:
Fill Pedestal Histograms (10³ events)
Fit Pedestal Histograms (Gaussian)
Output Pedestal Fits (Text/XML)
PEDESTALS (Text)
PEDESTALS (XML)

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. This process will be run twice w/ the Cal configured in both muon and flight gain. *genMuonPed.exe* needs 10^3 usable events, so the statistics requirement is not high. Nonetheless, multiple input filenames may be supplied in a newline delimited file.

9.1.2. Output

Pedestal results, including peak positions and widths, are stored in both an intermediate text file (for internal use) and an XML file which is exported to offline software system. 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:

9.2. 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 need to be in ROOT digi event format, which is currently automatically generated by the I&T pipeline.

For genCIDAC2ADC, we use element 201 of the CALF_COLLECT_CI_SINGLEX16 runs for the low energy linearity calibration and element 204 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 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.

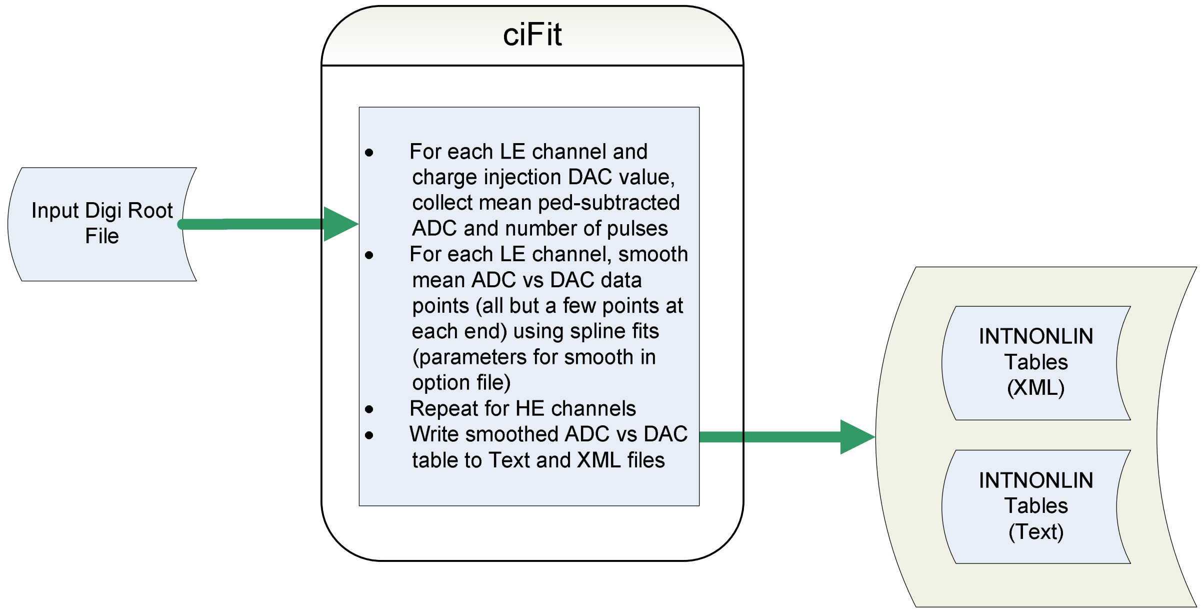

ciFit
Input Digi Root File
For each LE channel and charge injection DAC value, collect mean ped-subtracted ADC and number of pulses
For each LE channel, smooth mean ADC vs DAC data points (all but a few points at each end) using spline fits (parameters for smooth in option file)
Repeat for HE channels
Write smoothed ADC vs DAC table to Text and XML files
INTNONLIN Tables (XML)
INTNONLIN Tables (Text)

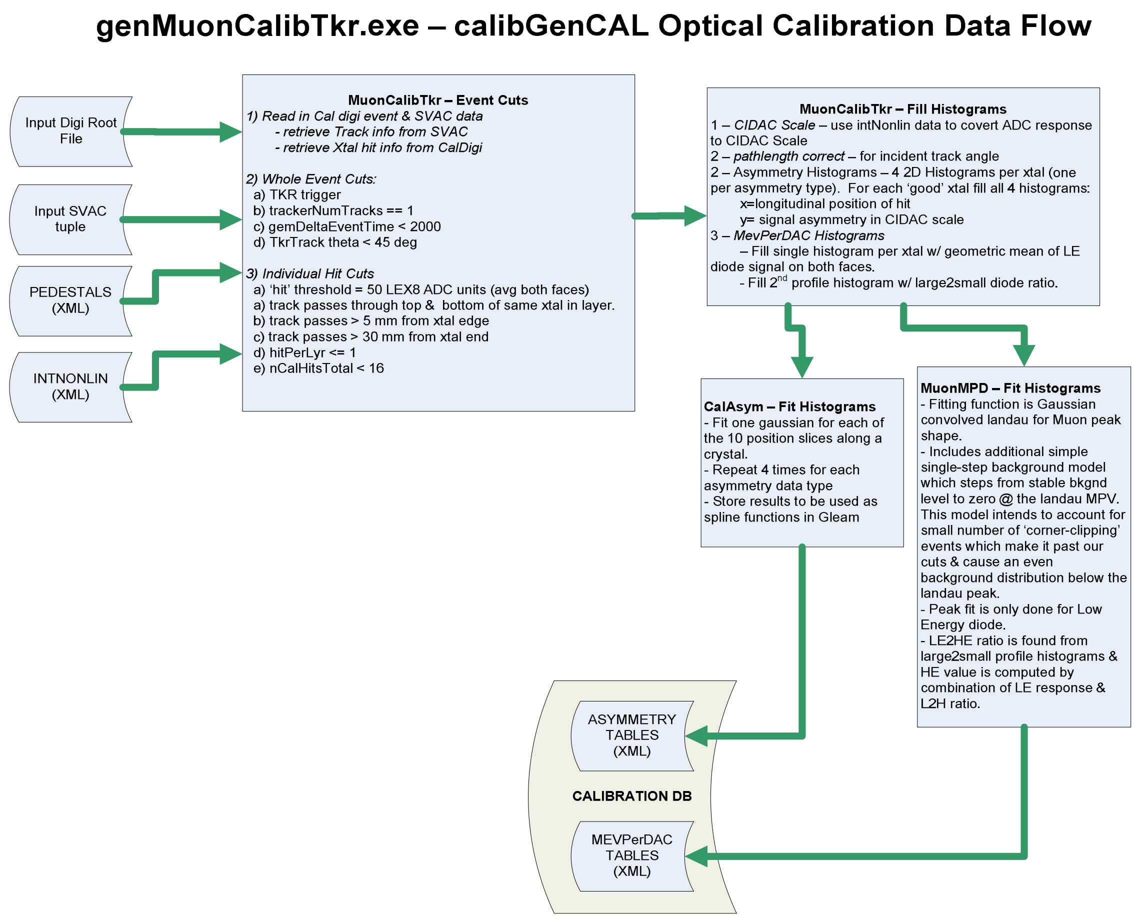
Figure 3 intNonlin Data Flow

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. Muon based Optical calibration (Asymmetry and MeVPerDAC)

9.3.1. Optical Calibration Data Flow


genMuonCalibTkr.exe – calibGenCAL Optical Calibration Data Flow

MuonCalibTkr – Event Cuts
1) Read in Cal digi event & SVAC data
- retrieve Track info from SVAC
- retrieve Xtal hit info from CalDigi

2) Whole Event Cuts:
a) TKR trigger
b) trackerNumTracks == 1
c) gemDeltaEventTime < 2000
d) TkrTrack theta < 45 deg

3) Individual Hit Cuts
a) 'hit' threshold = 50 LEX8 ADC units (avg both faces)
a) track passes through top & bottom of same xtal in layer.
b) track passes > 5 mm from xtal edge
c) track passes > 30 mm from xtal end
d) hitPerLyr <= 1
e) nCalHitsTotal < 16

MuonCalibTkr – Fill Histograms
1 – *CIDAC Scale* – use intNonlin data to covert ADC response to CIDAC Scale
2 – *pathlength correct* – for incident track angle
2 – Asymmetry Histograms – 4 2D Histograms per xtal (one per asymmetry type). For each 'good' xtal fill all 4 histograms:
x=longitudinal position of hit
y= signal asymmetry in CIDAC scale
3 – *MevPerDAC Histograms*
– Fill single histogram per xtal w/ geometric mean of LE diode signal on both faces.
- Fill 2nd profile histogram w/ large2small diode ratio.

CalAsym – Fit Histograms
- Fit one gaussian for each of the 10 position slices along a crystal.
- Repeat 4 times for each asymmetry data type
- Store results to be used as spline functions in Gleam

MuonMPD – Fit Histograms
- Fitting function is Gaussian convolved landau for Muon peak shape.
- Includes additional simple single-step background model which steps from stable bkgnd level to zero @ the landau MPV. This model intends to account for small number of 'corner-clipping' events which make it past our cuts & cause an even background distribution below the landau peak.
- Peak fit is only done for Low Energy diode.
- LE2HE ratio is found from large2small profile histograms & HE value is computed by combination of LE response & L2H ratio.

Figure 4 Optical Calibration Data Flow

The asymmetry and mevPerDAC calibration process is described in Figure 4 Optical Calibration Data Flow.

9.3.2. Asymmetry

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.

9.3.3. MeVPerDAC

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.

9.3.4. Input

9.3.4.1 Input File Type

genMuonCalibTkr.exe requires muon collection files with 4 range readout Cal data. The Cal should be configured in muon gain.

9.3.4.2 Statistics requirements

genMuonCalibTkr.exe needs very high statistics, multiple input filenames may be space delimited in the config file. *genMuonCalibTkr.exe* ideally will need 10^4 post-cut muon hits per crystal for asymmetry, $3x10^3$ for MeVPerDAC. The 16 tower LAT with 3e7 events will generate around half this much data. To get best available statistics, all usable input files should be included in the config file.

9.3.5. Output

9.3.5.1 Asymmetry 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 values 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 **:**

9.3.5.2 MeVPerDAC 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 :

9.4. 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. Tholdci xml file also stores charge-injection based adc pedestals.

The process of generating the threshold calibrations 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*.

9.5. 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 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.

9.5.1. LAC,FLE,FHE DAC setting algorithm

A. Summarize the xxx2adc.xml characterization files as a set of linear fit parameters (dacSlopes.xml file).

a. Convert target threshold energy to ADC units using muSlopes.xml file.

b. For FHE and FLE ADC values, apply bias correction.

c. Try linear fit for FINE range characterization data in region around threshold ADC value. If the FINE range produces a poor fit, linear fit the COARSE range characterization data in region around threshold ADC value. This produces fit parameters in ADC space.

d. Reverse the bias correction on fit parameters in ADC space for FHE and FLE.

e. Convert fit parameters from ADC space back to energy space using muSlopes.xml file

B. Now, to generate the settings simply use the dacSlopes.xml file fit parameters to calculate the nearest DAC setting which produces the required threshold in energy space. If the fit processing determined that the COARSE range DAC should be used, add 64 to the setting value.

9.5.2. ULD setting algorithm

A. Summarize the uld2adc.xml characterization files as a set of linear fit parameters (dacSlopes.xml file).

a. Do a linear fit for the entire COARSE range of ULD characterization data which falls below the saturation plateau.

b. Convert fit parameters and saturation value from ADC to energy space using muSlopes.xml file

B. The setting algorithm parameter is the minimum percentage of margin to allow between the ULD threshold and the range saturation value. The dacSlopes.xml saturation values are reduced by this amount, and the largest DAC setting which produces at least this margin in all three ranges (LEX8, LEX1, and HEX8) is chosen. Note that this most likely leaves a larger margin for the other two energy ranges because only one ULD setting is available.

10. calibGenCAL v4 Software Environment

10.1. Makefile

The entire calibGenCAL processing system is designed to be run from a Makefile supplied in calibGenCAL/cfg/Makefile. GNU Make is assumed. This system provides the following benefits.

- Extracts details of each individual application & it's command line interface.
- Understands all data interdependencies, ensuring that all required intermediate data products are automatically generated.
- Allows both high level processing, where numerous applications are controlled from a single command as well as facilitating generation of individual data products & intermediate data.
- Greatly eases analysis and debugging by automatically regenerating any data which depends on user-modified data.

Usage of the Makefile will be described in more detail in Section: 12.3.4 Discriminator DAC calibration

Cal discriminator DAC thresholds are characterized from measurements of calibDAC online test script. This documentation will cover processing from the filtered results of these charge injection calibration tests. Input is one characterization per DAC (FLE, FHE, ULD, DAC) for each of 16 towers. These files will be listed in a cfg file. Processing is repeated for flight and muon gain configurations. Characterization filename convention generally follows this template: `xxx_lac2adc_FM118_filtered.xml`.

Other inputs include:

- `muTrig.bias.xml` - calibrates bias between scintillation measured trigger thresholds and those measured w/ charge injection (such as calibDAC run).
- `muSlope.xml` – from optical calibration allows threshold dac characterization to be converted from adc scale to energy scale.

genDacSlopes.py fits slopes for the discriminator dacs against deposited energy scale. It takes as input the muSlope, bias, and threshold dac characterization files. It must be run separately for both muon gain and flight gain.

genDacSettings.py generates dac settings for given discriminator & intended energy from input *dacSlopes* file. It must be run once for each discriminator / gain setting combination.

tholdCIGen.py generates adc threshold calibration for a given instrument configuration. In addition to dacSlopes, dac settings, bias, IntNonlin, and MuSlope files… tholdCIGen also requires pedestal calibration files from calibDac test suite.

10.1.1.1Discriminator DAC data products list (FLE, FHE, ULD, LAC)

- `(MUON_GAIN|FLIGHT_GAIN).dacSlopes.xml` – dac slopes offline XML calibration file.
- `(MUON_GAIN|FLIGHT_GAIN).(lac|uld|fle|fhe).xml` – online dac settings XML files.
- `dacSlopes.val.(log|root) (lac|uld|fle|fhe).val.(log|root)` – validation files
- `(MUON_GAIN|FLIGHT_GAIN).tholdCI.(xml|val.log|val.root)` – offline tholdCI calibration file & validation outputs.

10.1.2. Onboard calibration

Onboard software, in particular the Onboard Filter (OBF) software will need Cal calibration information for purposes including event selection and data compression. calibGenCAL python scripts *genFlightPeds.py* and *genFlightGains.py* generate the C language .h header files for this purpose directly from the calPeds.xml & muSlope.xml offline calibration files.

10.1.2.1Onboard calibration data products

- cal_peds.h
- cal_gains.h

End 2 End Calibration Procedure Run Through The Makefile system should allow for users to work @ higher conceptual level without understanding all the processing details.

10.2. C++ based executables

10.2.1. 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.

10.2.2. 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:

```
fewtrell@iris01 $ genMuonPed.exe -h
Usage: '../src/genMuonPed [options] digiFilenames outputBasename '
Where:
        -h      --help  print usage info
        -t      --triggerCut    event filter 'ALL'|'PERIODIC'|'EXTERNAL' (default='PERIODIC')
        -e      --entriesPerHist        quit after all histograms have > n entries
                digiFilenames   text file w/ newline delimited list of input digi ROOT files
                outputBasename  all output files will use this basename + some_ext
```

Note that all calibGenCAL applications will output their command line interfaces via the '-h|--help' switch or simply when zero command line arguments are supplied.

10.3. Python scripts

10.3.1. Required software environment

calibGenCAL python scripts are developed with Python version 2.4 on both Linux and Windows. They make use of 3 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)

10.3.2. 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. CalXtalResponse : Usage of Cal calibration constants in Gleam

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

11.1. 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. $\mathrm{CIDAC}_{\mathrm{geom\ mean}} = \mathit{sqrt}(\mathrm{CIDAC}_{\mathrm{big\ plus}} * \mathrm{CIDAC}_{\mathrm{big\ minus}})$
        ii. $\mathrm{CIDAC}_{\mathrm{geom\ mean}} = \mathit{sqrt}(\mathrm{CIDAC}_{\mathrm{small\ plus}} * \mathrm{CIDAC}_{\mathrm{small\ minus}})$
        iii. $\mathrm{CIDAC}_{\mathrm{geom\ mean}} = \mathit{sqrt}(\mathrm{CIDAC}_{\mathrm{small\ equiv\ plus}} * \mathrm{CIDAC}_{\mathrm{small\ minus}})$
        iv. $\mathrm{CIDAC}_{\mathrm{geom\ mean}} = \mathit{sqrt}(\mathrm{CIDAC}_{\mathrm{small\ plus}} * \mathrm{CIDAC}_{\mathrm{small\ equiv\ minus}})$

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

       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($\text{CIDAC}_{\text{plus}}$/$\text{CIDAC}_{\text{minus}}$) as a function of position. P and M are as described in 6 above.

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

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

e. Compute energy as in 7.b above using the measured and estimated ends.

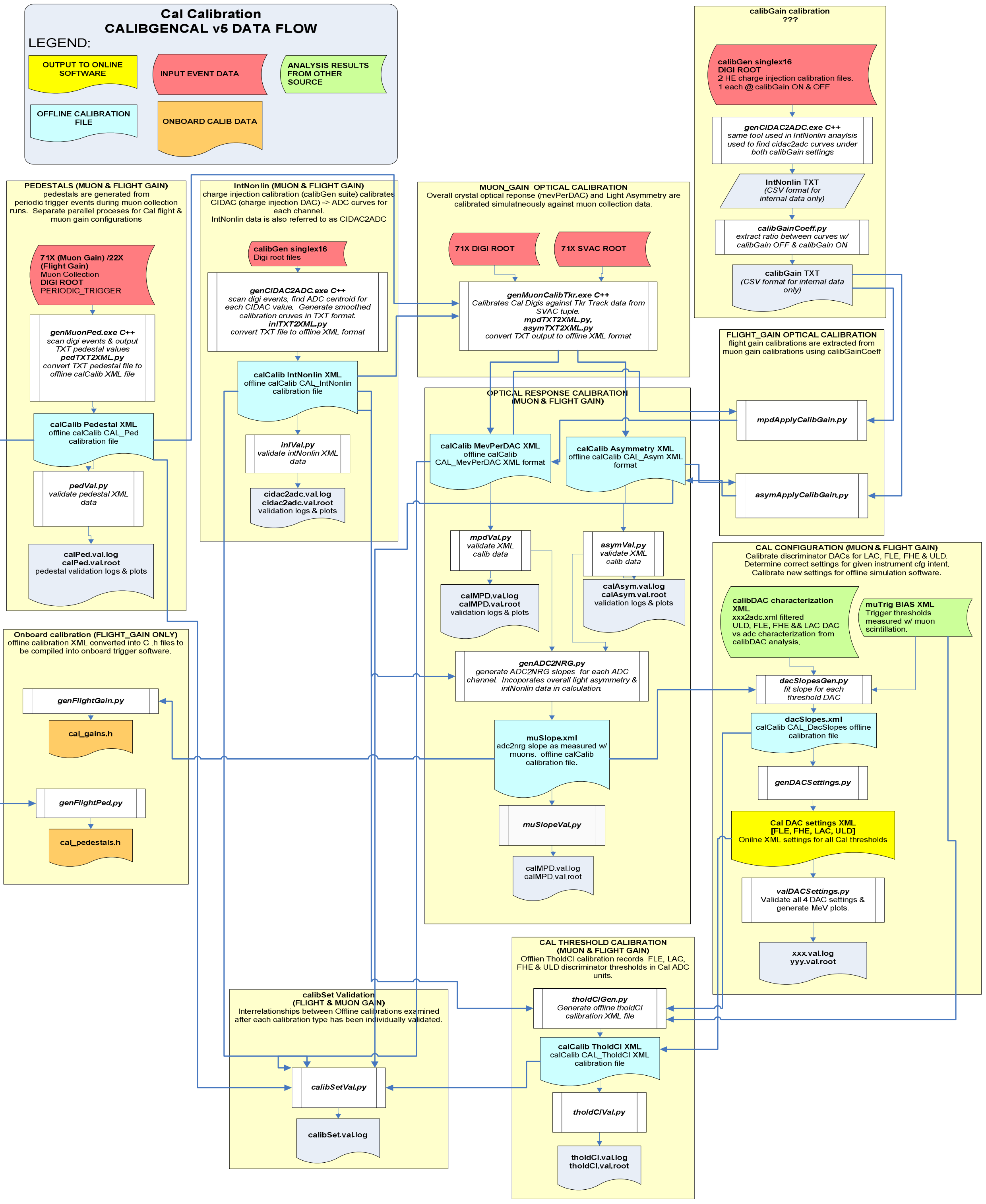
11.2. Single CDE Digitization Summary

This process is the inverse of the CDE Recon process described in Section 7. 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.

12. calibGenCAL Data Products

12.1. calibGenCAL Data Flow Diagram


Cal Calibration
CALIBGENCAL v5 DATA FLOW
LEGEND:
OUTPUT TO ONLINE SOFTWARE
INPUT EVENT DATA
ANALYSIS RESULTS FROM OTHER SOURCE
OFFLINE CALIBRATION FILE
ONBOARD CALIB DATA
PEDESTALS (MUON & FLIGHT GAIN)
pedestals are generated from periodic trigger events during muon collection runs. Separate parallel proceses for Cal flight & muon gain configurations
71X (Muon Gain) /22X (Flight Gain) Muon Collection DIGI ROOT PERIODIC_TRIGGER
genMuonPed.exe C++ scan digi events & output TXT pedestal values pedTXT2XML.py convert TXT pedestal file to offline calCalib XML file
calCalib Pedestal XML offline calCalib CAL_Ped calibration file
pedVal.py validate pedestal XML data
calPed.val.log calPed.val.root pedestal validation logs & plots
IntNonlin (MUON & FLIGHT GAIN)
charge injection calibration (calibGen suite) calibrates CIDAC (charge injection DAC) -> ADC curves for each channel.
IntNonlin data is also referred to as CIDAC2ADC
calibGen singlex16 Digi root files
genCIDAC2ADC.exe C++ scan digi events, find ADC centroid for each CIDAC value. Generate smoothed calibration cruves in TXT format. inlTXT2XML.py convert TXT file to offline XML format
calCalib IntNonlin XML offline calCalib CAL_IntNonlin calibration file
inlVal.py validate intNonlin XML data
cidac2adc.val.log cidac2adc.val.root validation logs & plots
MUON_GAIN OPTICAL CALIBRATION
Overall crystal optical reponse (mevPerDAC) and Light Asymmetry are calibrated simulatneously against muon collection data.
71X DIGI ROOT
71X SVAC ROOT
genMuonCalibTkr.exe C++ Calibrates Cal Digis against Tkr Track data from SVAC tuple. mpdTXT2XML.py, asymTXT2XML.py convert TXT output to offline XML format
calibGain calibration ???
calibGen singlex16 DIGI ROOT 2 HE charge injection calibration files. 1 each @ calibGain ON & OFF
genCIDAC2ADC.exe C++ same tool used in IntNonlin anaylsis used to find cidac2adc curves under both calibGain settings
IntNonlin TXT (CSV format for internal data only)
calibGainCoeff.py extract ratio between curves w/ calibGain OFF & calibGain ON
calibGain TXT (CSV format for internal data only)
FLIGHT_GAIN OPTICAL CALIBRATION
flight gain calibrations are extracted from muon gain calibrations using calibGainCoeff
mpdApplyCalibGain.py
asymApplyCalibGain.py
OPTICAL RESPONSE CALIBRATION (MUON & FLIGHT GAIN)
calCalib MevPerDAC XML offline calCalib CAL_MevPerDAC XML format
calCalib Asymmetry XML offline calCalib CAL_Asym XML format
mpdVal.py validate XML calib data
asymVal.py validate XML calib data
calMPD.val.log calMPD.val.root validation logs & plots
calAsym.val.log calAsym.val.root validation logs & plots
genADC2NRG.py generate ADC2NRG slopes for each ADC channel. Incoporates overall light asymmetry & intNonlin data in calculation.
muSlope.xml adc2nrg slope as measured w/ muons. offline calCalib calibration file.
muSlopeVal.py
calMPD.val.log calMPD.val.root
CAL CONFIGURATION (MUON & FLIGHT GAIN)
Calibrate discriminator DACs for LAC, FLE, FHE & ULD. Determine correct settings for given instrument cfg intent. Calibrate new settings for offline simulation software.
calibDAC characterization XML xxx2adc.xml filtered ULD, FLE, FHE && LAC DAC vs adc characterization from calibDAC analysis.
muTrig BIAS XML Trigger thresholds measured w/ muon scintillation.
dacSlopesGen.py fit slope for each threshold DAC
dacSlopes.xml calCalib CAL_DacSlopes offline calibration file
genDACSettings.py
Cal DAC settings XML [FLE, FHE, LAC, ULD] Onilne XML settings for all Cal thresholds
valDACSettings.py Validate all 4 DAC settings & generate MeV plots.
xxx.val.log yyy.val.root
Onboard calibration (FLIGHT_GAIN ONLY)
offline calibration XML converted into C .h files to be compiled into onboard trigger software.
genFlightGain.py
cal_gains.h
genFlightPed.py
cal_pedestals.h
CAL THRESHOLD CALIBRATION (MUON & FLIGHT GAIN)
Offlien TholdCI calibration records FLE, LAC, FHE & ULD discriminator thresholds in Cal ADC units.
tholdCIGen.py Generate offline tholdCI calibration XML file
calCalib TholdCI XML calCalib CAL_TholdCI XML calibration file
tholdCIVal.py
tholdCI.val.log tholdCI.val.root
calibSet Validation (FLIGHT & MUON GAIN)
Interrelationships between Offline calibrations examined after each calibration type has been individually validated.
calibSetVal.py
calibSet.val.log

12.2. Data products overview

12.2.1. Input files, ROOT, C++

Primary input file format for calibGenCAL are Cal digi ROOT event files. These files store the digital readout from the instrument for each event. Event files are generally processed in calibGenCAL with C++ applications. These applications use the CMT environment to link to ROOT libraries as well as GLAST custom ROOT based event data libraries.

calibGenCAL C++ applications will generally handle the tasks of:

- event looping
- histogram population
- histogram fitting.

Note: In general, output files names are based on input filenames, with appropriate filename extension modification.

12.2.2. Output files, XML, python

Post event processing in calibGenCAL is generally handled by python scripts. calibGenCAL python library is capable of reading and writing a wide array of XML file formats defined for GLAST online and offline software use.

calibGenCAL python scripts handle a wide array of tasks, notably:

- derived calibration quantity generation
- instrument settings generation
- data & file format validation
- trending, report generation
- file format conversion

12.2.3. Intermediary files

Intermediary files which are not intended for export outside the calibGenCAL package are sometimes stored in CSV, or delimited text tables. Often these files are output by C++ applications and they are converted to XML format by python scripts. calibGenCAL python library contains a full suite of conversions between internal CSV tables and the more official XML file formats.

12.2.4. Validation files

calibGenCAL python contains a suite of validation scripts for each calibGenCAL XML output. These scripts will generate both txt validation logs (xxx.val.log) as well as ROOT files w/ appropriate histograms & graphs (xxx.val.root).

12.3. Data products detailed

12.3.1. Pedestals

Two full sets of pedestals are generated, one for flight gain and one for muon gain configuration. *genMuonPed.exe* C++ application reads input digi event files, using either PERIODIC_TRIGGER, EXTERNAL_TRIIGER, or ZERO_SUPPRESSED event data to extract pedestals centroids & noise widths. C++ output is to txt file with root file histograms. Later phases of python scripts will convert to XML & validate.

12.3.1.1Pedestal file list

- `(MUON_GAIN|FLIGHT_GAIN).calPed.txt` – intermediary C++ output
- `(MUON_GAIN|FLIGHT_GAIN).calPed.root` –histograms
- `(MUON_GAIN|FLIGHT_GAIN).calPed.xml` – primary offline calibration output.
- `(MUON_GAIN|FLIGHT_GAIN).val.log` – validation log
- `(MUON_GAIN|FLIGHT_GAIN).val.root` – validation plots

12.3.2. IntNonlin (CIDAC2ADC)

Nonlinearity in ADC scales are calibrated against electronic charge injection DAC (CIDAC). *genCIDAC2ADC.exe* stores ADC vs DAC curves for each channel, extracting from digi event files: singlex16 data runs from calibGen test suite. Curves are smoothed and stored as a list of points, in practice, these curves will be interpolated using cubic splines. This process is repeated for both muon and flight gains. C++ output is to txt file. Later phases of python scripts will convert to XML & validate.

12.3.2.1 IntNonlin file list

- `(MUON_GAIN|FLIGHT_GAIN).cidac2adc.adcmean.txt` – raw adc means for each input CIDAC setting.
- `(MUON_GAIN|FLIGHT_GAIN).cidac2adc.txt` – intermediary C++ output, smoothed & pedestal subtracted.
- `(MUON_GAIN|FLIGHT_GAIN).cidac2adc.xml` – primary offline calibration output.
- `(MUON_GAIN|FLIGHT_GAIN). cidac2adc.val.log` – validation log
- `(MUON_GAIN|FLIGHT_GAIN).cidac2adc.val.root` – validation plots

12.3.3. Optical calibration (MeVPerDAC / Asymmetry)

Optical response in CsI crystals is calibrated against CIDAC scale. Optical calibration process depends upon pedestal and IntNonlin calibration. Ground calibration can only calibrate optical response in muon gain. Flight gain response is extrapolated from muon gain calibration using coefficients from charge injection calibration.

Muon portion of optical response calibration simultaneously calibrates overall optical efficiency of crystal (MeVPerDAC) along with Light Asymmetry. Input includes both digi event files from muon collections as well as SVAC tuple ROOT files from which Tracker Track information is extracted.

Note that flight gain IntNonlin is measured w/ calibGain setting on, which allows the CIDAC circuit to cover the full ADC range. Particle event data, however, is measured w/ calibGain setting off. An extra processing step will be required to match this data with the flight gain optical response coefficients.

The calibGain ratio is extracted by comparing singlex16 runs from calibGen suite which are similar except for different calibGain settings. The *genCIDAC2ADC.exe* C++ application is re-used for this purpose, but it's output is processed through *calibGainCoeff.py* script to extract the calibGain ratios.

The CalibGain On/Off ratio affects the entire ADC range linearly, and therefore has the same effect on overall bin width as an inverse scaling of the optical response constants. The latter method has been chosen for sake of simplicity. See following explanation:

- desired product is ratio of deposited energy to flight gain ADC:`(mev/adc_flight)`
- flight gain IntNonlin provides ratio: `(CIDAC_CALIBGAIN_ON/adc_flight)`
- muon gain mevPerDAC calibration provides: `(mev/CIDAC_CALIBGAIN_OFF)`
- calibGain analysis provides: `(CIDAC_CALIBGAIN_OFF/CIDAC_CALIBGAIN_ON)`
- it is clear from these ratios that
  - `mev/adc_flight=muongain_mpd*calibGainRatio*flight_inl`

Final mev / adc bin width is derived from these quantities by *genADC2NRG.py* . Output of this python script include MuSlope file which stores the adc2nrg slope as measured @ the muon peak for each Cal ADC channel. This allows for simple rough calculation of mev / adc which ignores non-linearity in the ADC scale. For many purposes, including threshold calibration and onboard trigger evaluation, this method is accurate enough & more desirable due to its computational & algorithmic simplicity.

12.3.3.1Optical Calibration file list

- `MUON_GAIN.muonOptical.calMPD.txt` – mev per dac constants from muon peak analysis
- `MUON_GAIN.muonOptical.calAsym.txt` – asymmetry constants from muon collection analysis.
- `MUON_GAIN.muonOptical.(calAsym|calMPD).xml` – primary offline calibration output.
- `MUON_GAIN.muonOptical.(calAsym|calMPD).val.(root|log)` – muon gain validation output.
- `calibGainOn.cidac2adc.txt, calibGainOff.cidac2adc.txt` - same TXT format as IntNonlin files.
- `calibGain.txt` – generated from cidac2adc curves with calibGainCoeff.txt

- `FILGHT_GAIN.muonOptical.(calAsym|calMPD).xml` – generated by applying calibGain ratio to muon gain optical coefficients. *asymApplyCalibGain.py* and *mpdApplyCalibGain.py* scripts are used.
- `FLIGHT_GAIN.muonOptical.(calAsym|calMPD).val.(root|log)` – flight gain validation output.
- `(FLIGHT_GAIN|MUON_GAIN).muSlope.xml` – final mev / adc bin width
- `(FLIGHT_GAIN|MUON_GAIN).muSlope.val.(log|root)` – bin width validation.

12.3.4. Discriminator DAC calibration

Cal discriminator DAC thresholds are characterized from measurements of calibDAC online test script. This documentation will cover processing from the filtered results of these charge injection calibration tests. Input is one characterization per DAC (FLE, FHE, ULD, DAC) for each of 16 towers. These files will be listed in a cfg file. Processing is repeated for flight and muon gain configurations. Characterization filename convention generally follows this template: `xxx_lac2adc_FM118_filtered.xml`.

Other inputs include:

- `muTrig.bias.xml` - calibrates bias between scintillation measured trigger thresholds and those measured w/ charge injection (such as calibDAC run).
- `muSlope.xml` – from optical calibration allows threshold dac characterization to be converted from adc scale to energy scale.

genDacSlopes.py fits slopes for the discriminator dacs against deposited energy scale. It takes as input the muSlope, bias, and threshold dac characterization files. It must be run separately for both muon gain and flight gain.

genDacSettings.py generates dac settings for given discriminator & intended energy from input *dacSlopes* file. It must be run once for each discriminator / gain setting combination.

tholdCIGen.py generates adc threshold calibration for a given instrument configuration. In addition to dacSlopes, dac settings, bias, IntNonlin, and MuSlope files… tholdCIGen also requires pedestal calibration files from calibDac test suite.

12.3.4.1Discriminator DAC data products list (FLE, FHE, ULD, LAC)

- `(MUON_GAIN|FLIGHT_GAIN).dacSlopes.xml` – dac slopes offline XML calibration file.
- `(MUON_GAIN|FLIGHT_GAIN).(lac|uld|fle|fhe).xml` – online dac settings XML files.
- `dacSlopes.val.(log|root) (lac|uld|fle|fhe).val.(log|root)` – validation files
- `(MUON_GAIN|FLIGHT_GAIN).tholdCI.(xml|val.log|val.root)` – offline tholdCI calibration file & validation outputs.

12.3.5. Onboard calibration

Onboard software, in particular the Onboard Filter (OBF) software will need Cal calibration information for purposes including event selection and data compression. calibGenCAL python scripts *genFlightPeds.py* and *genFlightGains.py* generate the C language .h header files for this purpose directly from the calPeds.xml & muSlope.xml offline calibration files.

12.3.5.1Onboard calibration data products

- cal_peds.h
- cal_gains.h

13. End 2 End Calibration Procedure Run Through

13.1. Makefile

The calibGenCAL makefile is stored in $CALIBGENCALROOT/cfg/Makefile

13.1.1. Software prerequisites

Most of these prerequisites will be available on a properly configured GLAST software installation.

- GLAST LAT offline software distribution (BeamtestRelease, GlastRelease, EngineeringModel)
- GLAST offline software CMT environment enabled. ('cmt config' or equivalent)
  - o this sets up needed PATH, PYTHONPATH && PKGPATH environment variables.
- python 2.4
  - o 4suite XML python package
  - o Numeric python package
- gnu make

13.1.2. Running make

In general, make procedure involves 3 stages:

1. Collection of data prerequisites
2. Population of make & cfg files
3. Running make tool

The default target for make will allow one to build all calibGenCAL products simply by typing 'make' (assuming they have specified all required inputs).

It is possible, however, to build any subset of these products by using the intermediate targets as long as the appropriate input files have been specified.

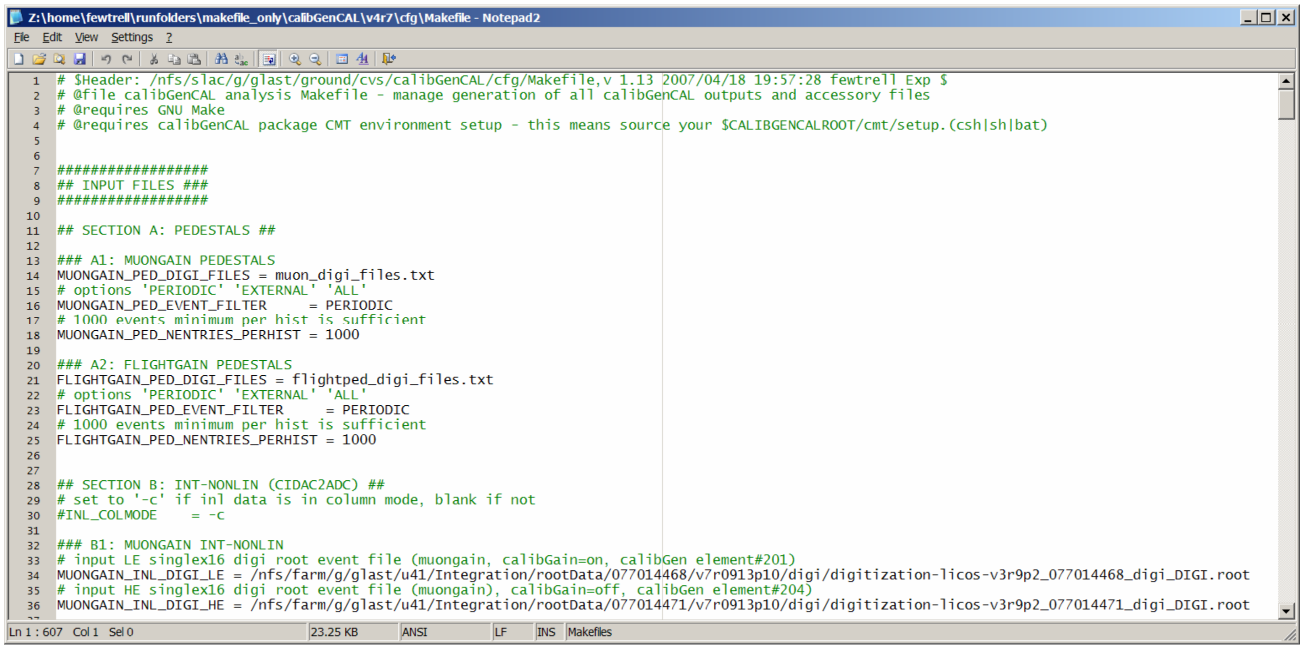
13.1.3. Data prerequisites

Mo

- muon gain pedestals
  - muon gain event digi files with either periodic trigger or external trigger or 4 range non-zero suppressed output.
- flight gain pedestals
  - flight gain event digi files
- IntNonlin
  - calibGen suite singlex16 digi files
- Optical calibration
  - all requirements for muon gain pedestals and intNonlin
  - muon collection files (muon gain)
- Threshold DAC settings & calibration
  - all requirements for optical calibration
  - ‘filtered’ calibDAC characterization XML
  - muTrig fle bias XML
  - calibDAC CI pedestal XML

13.1.4. *Specifying input files & configuration*

Most of the input files & software options can be specified in the Makefile itself. Begin by copying the default Makefile from calibGenCAL/cfg/Makefile directory.


Z:\home\fewtrell\runfolders\makefile_only\calibGenCAL\v4r7\cfg\Makefile - Notepad2
File Edit View Settings ?
Ln 1 : 607 Col 1 Sel 0
23.25 KB
ANSI
LF
INS
Makefiles

```
# $Header: /nfs/slac/g/glast/ground/cvs/calibGenCAL/cfg/Makefile,v 1.13 2007/04/18 19:57:28 fewtrell Exp $
# @file calibGenCAL analysis Makefile - manage generation of all calibGenCAL outputs and accessory files
# @requires GNU Make
# @requires calibGenCAL package CMT environment setup - this means source your $CALIBGENCALROOT/cmt/setup.(csh|sh|bat)


##################
## INPUT FILES ###
##################

## SECTION A: PEDESTALS ##

### A1: MUONGAIN PEDESTALS
MUONGAIN_PED_DIGI_FILES = muon_digi_files.txt
# options 'PERIODIC' 'EXTERNAL' 'ALL'
MUONGAIN_PED_EVENT_FILTER      = PERIODIC
# 1000 events minimum per hist is sufficient
MUONGAIN_PED_NENTRIES_PERHIST = 1000

### A2: FLIGHTGAIN PEDESTALS
FLIGHTGAIN_PED_DIGI_FILES = flightped_digi_files.txt
# options 'PERIODIC' 'EXTERNAL' 'ALL'
FLIGHTGAIN_PED_EVENT_FILTER      = PERIODIC
# 1000 events minimum per hist is sufficient
FLIGHTGAIN_PED_NENTRIES_PERHIST = 1000


## SECTION B: INT-NONLIN (CIDAC2ADC) ##
# set to '-c' if inl data is in column mode, blank if not
#INL_COLMODE     = -c

### B1: MUONGAIN INT-NONLIN
# input LE singlex16 digi root event file (muongain, calibGain=on, calibGen element#201)
MUONGAIN_INL_DIGI_LE = /nfs/farm/g/glast/u41/Integration/rootData/077014468/v7r0913p10/digi/digitization-licos-v3r9p2_077014468_digi_DIGI.root
# input HE singlex16 digi root event file (muongain), calibGain=off, calibGen element#204)
MUONGAIN_INL_DIGI_HE = /nfs/farm/g/glast/u41/Integration/rootData/077014471/v7r0913p10/digi/digitization-licos-v3r9p2_077014471_digi_DIGI.root
```

Figure 5 calibGenCAL Makefile screenshot

The first section of the makefile is marked “INPUT FILES” and is intended to be user edited. Populate the appropriate fields w/ the input files for your calibration task.

13.1.4.1 *Secondary configuration files*

Some applications which require a large number of input files have individual configuration files. In these cases, the cfg filename will be specified in the make file & the secondary configuration file must be populated as well.

These secondary configuration files include newline delimited event file lists for *genMuonPed.exe* and *genMuonCalibTracker.exe* as well as ini style configuration files for *genDACSlopes.py* and *genTholdCI.py*.

Example cfg files can be found in $CALIBGENCALROOT/cfg/

13.1.5. Individual make targets.

The calibGenCAL Makefile provides numerous ‘targets’ for generating individual data products as well as logical groupings. The default, or ‘all’ target will generate the full set of calibGenCAL outputs and requires only that the user enter ‘make’ or ‘make all’ into the command-line after specifying the data prerequisites.

In general the target names follow the following pattern.

For single files:

`<gainsetting>_<calibration>_<filetype> – example, “muongain_ped_xml”, “flightgain_inl_val”.`

For file groups:

`<gain_setting>_<calibration>` - example `“muongain_muopt”` or `“flightgain_dacsettings”` which will generate all related files, including TXT, XML and validation for the given calibration type.

Here is a table of useful targets for individual data products & sub-groups. To invoke any one, simply type ‘make <target>’ into the command line from the same directory as the Makefile.

TARGET NAME	**DATA PRODUCT**
all	all calibGenCAL data products
flightgain_calib	all flight gain calibrations
muongain_calib	all muon gain calibrations
inl, muongain_inl, flightgain_inl	IntNonlin calibrations
calib_gain	calibGain coefficients
ped, muongain_ped, flightgain_ped	pedestal calibrations
muongain_muopt, flightgain_muopt	optical calibrations (mevPerDAC & asymmetry)
onboard_calib	onboard calibrations
muslope, flightgain_muslope, muongain_muslope	muSlope calibrations
dacslopes, flightgain_dacslopes,	dacSlope calibrations

muongain_dacslopes	
tholdci, flightgain_tholdci, muongain_tholdci	tholdCI threshold calibrations
dacsettings, muongain_dacsettings, flightgain_dacsettings	Discriminator DAC settings