How to setup up and generate the ACD calibration constants

David M. Green

August 4, 2017

1 Introduction

I'm intending this to be a short, quick, and informal memo on how to setup and generate the ACD calibration constants used in long-term trending of the stability of the ACD. I will also detail the ACD subsystem, each of the ACD calibration constants, and how the calibration constants are generated and used in the reconstruction and simulation of data related to the ACD. I'm not planning outright to go into explicit detail (one never does) but I will provide an adequate background in these topics. There are a number of calibration constants, I will go over each in detail, but I want to note here the most important constants used for long-term trending are the low-rage pedestals and low-range gains.

2 The Fermi Large Area Telescope Anti-coincidence Detector

The Fermi LAT's Anti-coincidence detector (ACD) is the main subsystem for the removal of cosmic-rays from the γ-ray signal. The ACD consists of 89 plastic scintillating tiles and 8 plastic scintillating ribbons for a total of 97 elements. Each ACD element is a plastic scintillator of polyvinyl toluene (PVT). The 97 ACD elements are arranged on five sides with 5x5 tiles on the top, 3x5 on each side and an additional large tile on the bottom of each side for 'back-splash' reduction, and the ribbons covering the gaps between tiles to increase cosmic-ray detection. This arrangement can be seen in Figure 1.

Each ACD element has two photomultiplier tubes (PMT) which have a number of calibration constants which characterize the response of the PMT to signal. The ACD calibration constants which characterize the response of the each PMT are: low pedestals, low range gains, coherent noise, range cross-over, high range pedestals, carbon peak, and high range calibration. There are additional calibration constants which are related to the veto and CNO triggers. The veto and CNO set point calibrations constants are used to determine when the veto and CNO fire. The veto and CNO threshold fits relate the set points to the FREE settings. The PMT's have two ranges: a linear low range used to read signal from protons, electrons, and helium, and a non-linear high range used for signal from carbon, nitrogen, oxygen, and other heavier cosmic-rays.

There are also a few common terms used in the generation of the calibration constants. Pulse height amplitude (PHA) is amplitude of the analog signal in the PMT or essentially the raw signal from the PMT. Minimum ionizing particle (MIPs) a particles whose energy loss rate is close to the minimum of the ionization energy loss curve. The ionization energy loss curve is dependent on the type of material and the amount of material cross but for the ACD typically refers to 1.9 MeV. Let's go over each calibration constant in a bit more detail and how the calibration constants are generated.

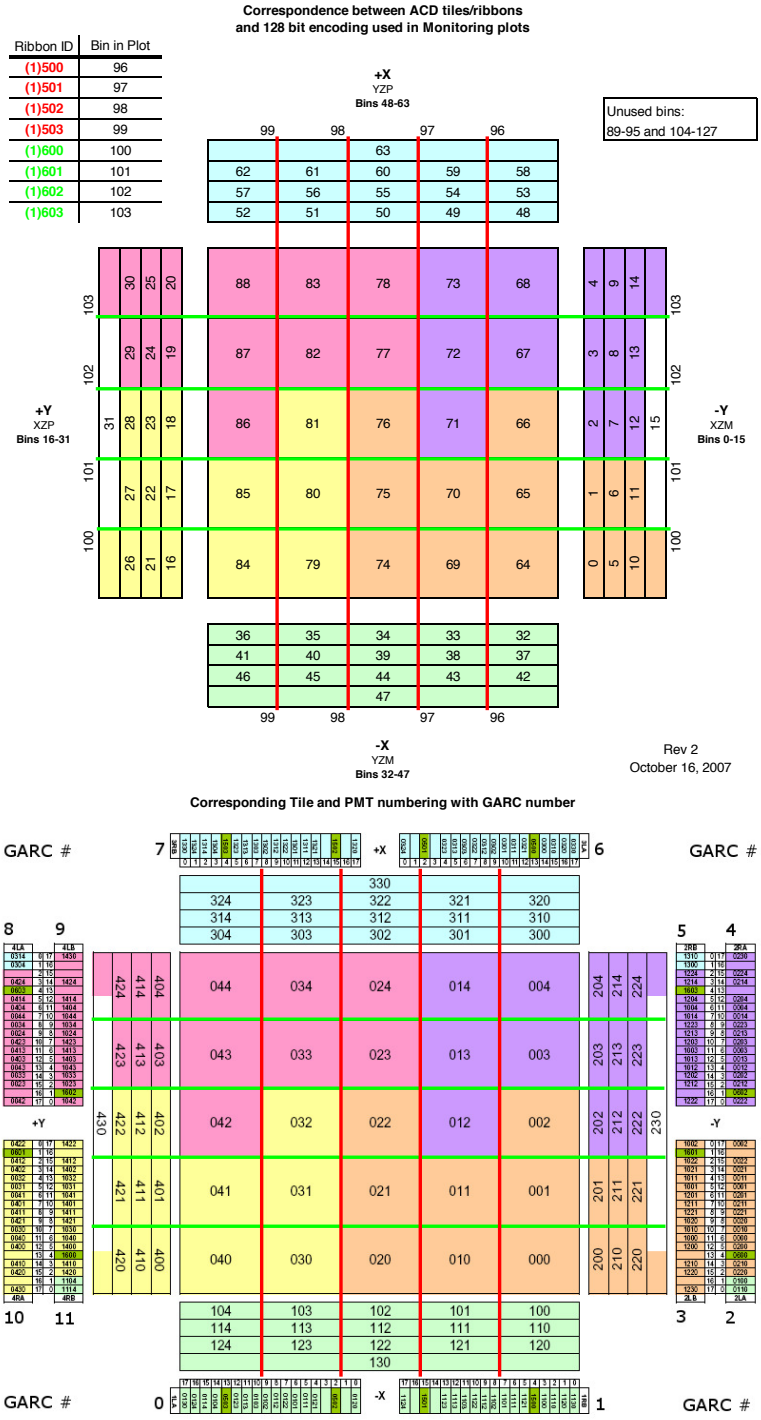

Figure 1: Map of ACD elements with identities used for generation of ACD calibration constants and datamon. Additionally the associating between the FREE board and the PMT are shown.

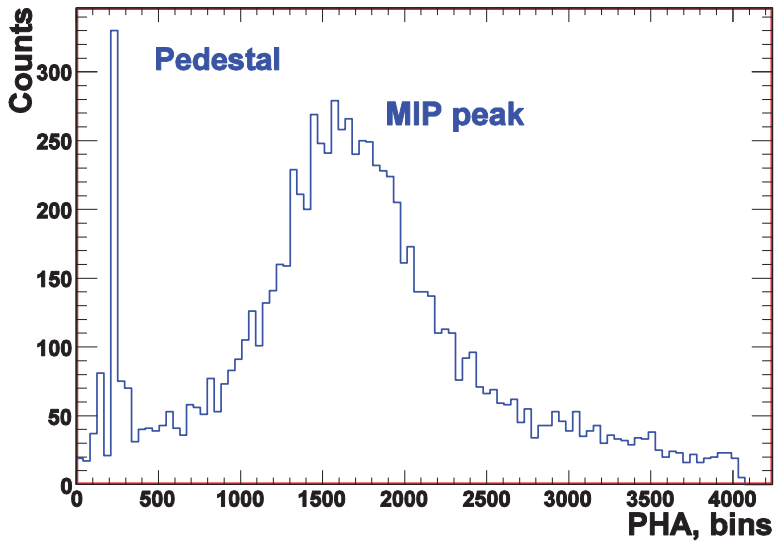

Figure 2: The pedestal and MIP peaks seen in a single PMT in the low range for an ACD element.

2.1 Low Range Pedestal

The low range pedestal is the measure of the electrons base line in the PMT without any signal from cosmic-rays. Each PMT has a pedestal which much be subtracted from any signal to properly estimate how much energy was deposited in the ACD element. Low range pedestals are determined from using **DIGI** files and periodic triggers. Periodic triggers are all signals and no filters on the flight data for a short period of time every SOME NUMBER. **DIGI** files are used because they contain all of the data, both calibrated and uncalibrated, from the LAT including ACD PTM signal in units of PHA. An example of a pedestal can be seen in Figure 2 below 100 PHA and is dominate over the MIP peak. To find the value of the pedestals, we find the mean and RMS from period trigger events only in the low range of the PMT and reject signal that is more than 5 RMS from the mean and remeasure the mean and RMS. The low range pedestal only requires of run of data in order to complete. This process is done by the program `runPedestal.cxx`.

2.2 Low Range Gains

Low range gains is used convert signal in the low range from instrument units (PHA) to a a physical units of MIP equivalent units (MEQ). MEQ is the signal one would expect from a MIP passing through an ACD element which should then translate to a deposited energy of 1.9 MeV. This is critical for understanding how much energy is deposited in the ACD and for the rejection of cosmic-rays from the gamma-ray dataset. We measure the gain of the low range for PMTs by using low energy protons and measuring the MIP peak a seen in Figure 2 from events that deposit signal in the low range of the PMT. The peak is fit using a Gaussian distribution plus a first order polynomial and finding the mean and σ of the distribution. This calibration also requires a pedestal which is subtracted from all PHA signal. This calibration uses **RECON** data since it requires events to have passed several quality cuts, direction of the event for path length correcting the signal, and the ACD signal to be in units of PHA. The low range gain requires about 5 runs of data and a low range pedestal (2.1) in order to complete. This process is done by the program `runMipCalib.cxx`.

2.3 Coherent Noise

Coherent noise is an issue on the low range of PMTs where the read out noise is shaped by the previous event if the previous event was too close the current trigger window. This has the effect where the pedestal is increased or decreased by 20 PHA depending on how far away the previous event was from the beginning of the trigger window. The method is to collect signal in PHA vs time (using GemDeltaEventTime) for periodic triggers. The profile of the coherent noise if fit with

$$\texttt{Signal [PHA]} = Ae^{-t/\Gamma}\sin\left(0.054\,t+\delta\right). \tag{1}$$

A is the amplitude, Γ is the decay, t is the time, and δ is the phase and each are used in the calibration of the coherent noise. The coherent noise calibration only requires of run of **DIGI** data in order to complete. This process is done by the program `runNoiseCalib.cxx`.

2.4 High Range Pedestals

Similar to that described in 2.1 but for the high range instead of the low range. Unlike the low range pedestals, the high range pedestals cannot be determined from periodic triggers and requires light charge injection (LCI) to measure. LCI runs are take on annual intervals and require the loss of a data run. The same procedure is used to find the high range pedestals as discussed in 2.1. This once again uses **DIGI** files of LCI runs. The high range pedestal only requires of run of data in order to complete. This process is done by the program `runHighPed.cxx`.

2.5 Carbon Peaks

The carbon peaks are the analogous process of 2.2 to find the high range gain. Instead of using MIPs, we use minimum ionizing carbon cosmic-rays which should deposit 36 MEQ in an ACD element. This requires the use of **RECON** files since it further requires event selections in based on CAL information to find suitable carbon cosmic-ray events to perform the calibration. The carbon peak is fit using a Gaussian distribution where the mean and σ are used as the calibration constants. The carbon peaks requires about 5 days of data and a high range pedestal (2.4) in order to run. This process is done by the program `runCarbonCalib.cxx`.

2.6 Range cross-over

The range cross-over calibration measures the PHA value where the signal readout switches from low range to high range. This calibration is used in simulation to provide realistic PHA values. We then scan the low range histogram to find the highest low range PHA value and the high range histogram to find the lowest high range PHA value The method is to collect PHA for all events into two histogram, one for low range readout, one for high range. The range cross-over calibration requires five **DIGI** runs, a low range pedestal (2.1), and a high range pedestal (2.4) in order to run. This process is done by the program `runRangeCalib.cxx`.

2.7 High Range Calibration

The high range calibration provides the signal scale for events read out in the high range of the ACD. This calibration is used in reconstruction to express the raw PHA values in terms of MIP equivalent signals and eventually in MeV. The reason one needs to use the high range calibration is because the

high range is non-linear and a saturation effect occurs at 2000 PHA. The value of high range PHA per MIP is determined and this is used in the following equation to convert to signal in units of MEQ:

$$\texttt{Signal[MEQ]} = \frac{2000\ \texttt{PHA} \times (\texttt{Signal[PHA]} - \texttt{Ped[PHA]})}{\texttt{gain [PHA/MEQ]} \times (2000\ \texttt{PHA} - \texttt{Signal[PHA] + Ped[PHA]})} \quad (2)$$

It should be noted that the saturation is set to 2000 PHA but this is not representative of the actual saturation for each PMT but since higher charged cosmic-rays are easily vetoed, this is not considered an important issue. The high range calibration does not use flight data but high range pedestals (2.4), carbon peaks (2.5), and range crossover (2.6) in order to run. This process is done by the program `runHighRangeCalib.cxx`.

3 The Setup

The first step is to setup some environment variable. In your cshrc or bash file add the two following lines.
For cshrc:

```
setenv GLASTROOT /afs/slac.stanford.edu/g/glast
source ${GLASTROOT}/ground/scripts/group.cshrc
```

For bash:

```
export GLASTROOT=/afs/slac.stanford.edu/g/glast
source ${GLASTROOT}/ground/scripts/group.sh
```

This sets up a number of environment variables which are crucial for generating the calibration constants and storing them in the correct locations. This also sets up the CVS application and allows you 'checkout' software for `GlastRelease` (GR), which is used to generate the calibration constants.

From there, the user needs to setup a custom version of `GlastRelease`. Make a directory called `releases` in your AFS disk space. This is where your checkout version of GR is going to live. In this directory make a folder called `GR-20-09-10`, this version of `GlastRelease` has been validated to work with `calibGenACD`.

Next make a directory called `workdir`. This will help in the future if you need a directory to work, run macros, store files, and various things that will not be compiled with scons. Also make a directory called `python`. The next step is make a config file for the users version of `GlastRelease`. Here is template for the c-shell file config.csh:

```
#!/bin/csh

setenv SCONS_REL 20-09-10
setenv SCONS_VARIANT redhat6-x86_64-64bit-gcc44
setenv SCONS_VERSION Optimized
setenv SCONS_TOTAL ${SCONS_VARIANT}-${SCONS_VERSION}
setenv SCONS_BUILDS /nfs/farm/g/glast/u52/ReleaseManagerBuild
setenv GLAST_EXT_BASE /afs/slac/g/glast/ground/GLAST_EXT
setenv GLAST_EXT ${GLAST_EXT_BASE}/${SCONS_VARIANT}
setenv PARENT ${SCONS_BUILDS}/${SCONS_VARIANT}/${SCONS_VERSION}/GlastRelease/${SCONS_REL}
setenv RELEASE /afs/slac/g/glast/users/username/releases/GR-${SCONS_REL}
setenv INST_DIR $RELEASE
```

```
setenv BASE_DIR $PARENT
setenv CALIBGENACDROOT $RELEASE/calibGenACD/
setenv FACILITIESROOT $PARENT/facilities
setenv RDBMODELROOT $PARENT/rdbModel
unsetenv PYTHONPATH
unsetenv LD_LIBRARY_PATH
unsetenv DYLD_LIBRARY_PATH
```

and for bash file config.sh:

```
#!/bin/sh

export SCONS_REL=20-09-10
export SCONS_VARIANT=redhat6-x86_64-64bit-gcc44
export SCONS_VERSION=Optimized
export SCONS_TOTAL=${SCONS_VARIANT}-${SCONS_VERSION}
export SCONS_BUILDS=/nfs/farm/g/glast/u52/ReleaseManagerBuild
export GLAST_EXT_BASE=/afs/slac/g/glast/ground/GLAST_EXT
export GLAST_EXT=${GLAST_EXT_BASE}/${SCONS_VARIANT}
export PARENT=${SCONS_BUILDS}/${SCONS_VARIANT}/${SCONS_VERSION}/GlastRelease/${SCONS_REL}
export RELEASE=/afs/slac/g/glast/users/username/releases/GR-${SCONS_REL}
export INST_DIR=$RELEASE
export BASE_DIR=$PARENT
export CALIBGENACDROOT=$RELEASE/calibGenACD/
export FACILITIESROOT=$PARENT/facilities
export RDBMODELROOT=$PARENT/rdbModel
unset PYTHONPATH
unset LD_LIBRARY_PATH
unset DYLD_LIBRARY_PATH
```

replacing `username` with the user's username. Make the config files executable using `chmod`

```
chmod +x config.csh
```

The next thing you need is an script that compiles code using scons. In a file called `scons_make` put:

```
scons -i -C ${PARENT} --with-GLAST-EXT=${GLAST_EXT} --duplicate=soft-copy \
--exclude=workdir --supersede=${RELEASE} --rm --compile-opt $*
```

What this is doing is running the program scons, creating a soft copy against the files located in `$PARENT` creating the appropriate links in `$RELEASE` and ensuring that when running programs it will use the locally checked out files in `$RELEASE` and everything else in `$PARENT`. Make the file executable:

```
chmod +x scons_make
```

With all of the files made, source the config file:

```
source config.csh or source config.sh
```

This will setup proper environment variables and should always be run when using applications in GR-20-09-10, generating calibration constants, or editing checked out packages.

The next step is checkout two packages via CVS in GlastRelease: calibGenACD and mootCore. CVS is a code repository and management system, think of it like a precursor to git. The base files live in $PARENT and you should NOT try to edit the files in the $PARENT directory and you will probably not be able to anyway. In the GR-20-09-10 directory, run the following commands:

```
cvs checkout calibGenACD
cvs checkout mootCore
```

This will checkout the packages with uncompiled source files. You can edit these files in your $RELEASE directory, compile using scons, and run the applications locally.

Before running scons_make, you need to make sure scons produces a python library file for mootCore. In the mootCore directory there should be a file called SConscript. Open it in your text editor of choice and change the line says:

```
if 'CHS' in progEnv.Dictionary()['CPPDEFINES']:
```

to

```
if True:
```

This will run swig and make the appropriate python library file for mootCore. Next run scons_make to compile the files.

```
./scons_make
```

If this worked you will only have two failed nodes:

```
scons: printing failed nodes
/afs/slac/g/glast/users/username/releases/GR-20-09-10/calibGenACD/build/
redhat6-x86_64-64bit-gcc44-Optimized/apps/runOverlayPedestal.o
/afs/slac/g/glast/users/username/releases/GR-20-09-10/calibGenACD/build/
redhat6-x86_64-64bit-gcc44-Optimized/runOverlayPedestal
```

This is fine, runOverlayPedestal isn't used in the generation of the ACD calibration constants.

In order to run the python programs in Section 4.2 we need python to talk to the moot database which generates information on flight **DIGI**, **RECON**, and **MERIT** files. Find the file mootCore/build/redhat6-x86_64-64bit-gcc44-Optimized/src/py_mootCore.py in your $RELEASE folder copy it to $RELEASE/python. Everything should work when running ParseFileListNew.py in Section 4.2.

Next you need to update the calibGenACD source files to work with Pass 8. This is a bit janky because I never committed my changes to the calibGenACD code to the CVS repository (I know, I am planning to do this in the future and when I do I will edit this memo to reflect those changes). Copy all of .cxx and .h files in /nfs/farm/g/glast/u/damgreen/releases/GR-20-09-10_TEST/calibGenACD/src/ to your $RELEASE/calibGenAD/src/ folder. Next rerun scons_make and make sure everything compiles. Finally, copy $RELEASE/bin/redhat6-x86_64-64bit-gcc44-Optimized/_setup.csh or $RELEASE/bin/redhat6-x86_64-64bit-gcc44-Optimized/_setup.sh to $RELEASE. Whenever you need use your custom version of GlastRelease you need to source both your config file and your _setup file in that order. Now you should be able to generate ACD calibration constants using Pass 8 and this will be explained in Section 4.2.

4 Running the Program

As a reference, the long term trending of calibration constants is kept on this web-page:
`http://www.slac.stanford.edu/exp/glast/acd/calib_reports/`
which accesses this location within afs:
`/afs/slac.stanford.edu/g/glast/ground/releases/monitor/ACD/FLIGHT`
which is related to the `$LATMonRoot` from Section 3.

4.1 ParseFileListNew

Running the program is very simple. Whenever you need use your custom version of `GlastRelease` you need to source both your config.c(sh) file and your _setup.(c)sh file in that order in your `$RELEASE` folder. The first step is get into `calibGenACD/python` where the python programs that generate the calibration constants are stored. Once there, the first step is run `ParseFileListNew.py`. `ParseFileListNew.py` finds lists of **DIGI**, **RECON**, and **MERIT** files and output two documents a list and table of the relevant information for each data-type. You need to produce the table of runs for both **DIGI** and **RECON** file types.

```
python ParseFileListNew.py DIGI
python ParseFileListNew.py RECON
```

Make sure you have four files that should look like `DIGI_DATE.list`, `DIGI_DATE.table`, `RECON_DATE.list`, and `RECON_DATE.table` where date is the current date in the year, month, day format (eg August 1st, 2017 is 170801).

Check to make sure these files are populated. You should also to check to see if there are enough runs in both .table files for `DIGI` and `RECON` in order to run the bi-weekly calibrations. For instance, if it is week 500 and there are only 2 `RECON` files, then wait a day or so for files to finish processing and pass datamon. You will need to re-run `ParseFileListNew.py` to re-scan for the new `DIGI` and `RECON` files.

These files access the moot query database, so if that is not working from Section 3, it needs to be fixed to proceed. Go into the `mootCore`

```
swig -python -I/$PARENT/facilities -I/$PARENT/rdbModel -I/$RELEASE/mootCore \
-c++ py_mootCore.i
```

4.2 AcdWeeklyReport

`AcdWeeklyReport.py` is the main program that accesses the other programs to generate calibration constants. It has two modes `run` and `trend`. `run` mode generates the calibration constants in the `CALIBTYPES` array at the top of the file. `run` mode runs `AcdReportRun.py` for each calibration type and submits jobs to the batch farm. `AcdReportRun.py` builds the list files used by the individual apps which `AcdReportUtil.py` runs. `AcdReportUtil.py` is the program that moves the completed files to the correct locations in `$LATMonRoot` and updates the html pages on the calib reports website. The syntax for using `AcdWeeklyReport.py` in `run` mode is:

```
python AcdWeeklyReport.py 'run' -w 500 DIGI_170801.table RECON_170801.table
```

where the `-w` is for the week of data you wish use in generation of calibration constants. As of writing this the latest week is 476. You have to provide the `DIGI_DATE.table` first and the `RECON_DATE.table` second or else it will fail.

For the first time, I would recommend commenting out everything after `os.mkdir(toDir)` in `AcdReportUtil.py` to test to make sure the ACD calibration constants are generated correctly. Each program will generate a number of files including pdfs, txt, xmls, and root files. I would also recommend checking against all ready run weekly calibration reports on the ACD calib web-page and see if they agree. If everything checks out, then uncomment out everything below `os.mkdir(toDir)` in `AcdReportUtil.py` and rerun the weekly report.

The second mode is `trend` which runs `AcdReportTrend.py` for each calibration type. To use this mode, simply run:

```
python AcdWeeklyReport.py 'trend'
```

This produces a trending plot each calibration constant listed in Section 2. Each calibration in `$LATMonRoot/ACD/FLIGHT/` has a calibs.lst which is appended when a new week of calibration constants is generated. Since the calibration constants are checked every two weeks, the user has to update each calibs.lst and place a 'none' in the weeks that the calibration constants are not checked. The trending finds the changes to calibration constants over the weeks to a reference calibration and produces the trending plots. Once again, comment out the last line of `AcdReportTrend.py` and make sure the trending plots make sense with regard to what is on the ACD calib web-page. If everything checks out, uncomment and rerun AcdWeeklyReport.py and this will move the trending plots and pages to appropriate location and update the ACD calibration web-page (you might need to refresh the web-page a few times to see the changes).

That's it! Like stated earlier, the calibration constants should be checked every two weeks. The best practice is to check the calibration constants every two weeks