Crash Course on Working with HAWC Event Data With a Special Emphasis on Simlation Weighting

John Pretz

Aug 24, 2014

Abstract

This memo outlines how to work with HAWC event data, particularly using the weights in simulation. This is a current snapshot and the reader should note we anticipate changes to address the large number of unnecessarily diverse formats. In particular: There are planned re-writes to the weighting code. The data formats are being re-thought to facilitate the development of new parameters. And a new step involving simulation of the FEB is planned. Nevertheless, the weighting required to model physical fluxes can be a bit confusing especially for new users. If it turns out that this document is useful to people, future expansions are possible.

1 Weighting

The main challenge in working with HAWC simulation is the weighting involved to reproduce physical fluxes. In order to efficiently generate simulation, four unphysical biases are introduced during simulation generation.

First, the photon and hadron simulation is generated on a E^{-2} differential energy spectrum. Physical fluxes are generally softer than this and, in the case of photon spectra, can cut off at multi-TeV energies. The artificially-hard spectrum at generation guarantees that we will have sufficient simulation of extremely high-energy events for whatever studies we want. It's worth noting that the time required for the simulation of an event is roughly proportional to the energy of the event. The E^{-2} thrown spectrum means we spend roughly equal amounts of time in each decade of energy simulated. There is really not much benefit to further hardening the simulated spectrum.

The second bias introduced is a radial bias. We throw events from a distribution that's flat in R, the distance of the air shower core from the center of the simulated geometry. A natural distribution would mean the number of showers thrown at a given R would be proportional to R, the differential area of the detector at a distance R. Since our throwing distribution is flat in R, this means we have artificially too many events thrown at low R, near the detector. The rationale behind this bias is that, while we want to probe the high-R behavior of the detector (e.g. high-energy showers at high R with low-energy high-pt subshowers that strike the detector are an important background) we don't want to devote extensive simulation time to high-R events that, by and large, will be removed by our analysis cuts.

The third bias introduced is the relative abundance of cosmic ray populations. Hydrogen, Helium and the higher-Z cosmic rays occur in specific relative abundances. During simulation, we throw isotopes independently and make no effort to maintain the natural relative abundances. Since each isotope occurs with a different spectrum the issue is co-mingled with the simulated spectral index. Instead, we "fix" the relative abundances along with the spectral index with weighting.

Finally, the last bias introduced is specific to the simulation of gamma rays. The gamma -ray sources we are interested in are primarily point sources. They transit through our field of view spending an amount of time at each θ and ϕ in the detector given by the declination of the source. We throw gamma rays as though they were isotropically arriving at the detector. Transiting photon sources are handled through weighting by "fixing" the distributions.

A prior memo[1] describes the weighting much more completely and describes how the weights may be calculated. The reader is directed there for more complete discussion.

2 Data Formats

Figure 1 shows the main data formats and processing steps for an analysis of the HAWC triggered data. Eight different data formats are shown. Many of the file formats are XCDF, a compact bit-packed data format written specifically for HAWC. We discuss working with XCDF files in a subsequent section.

- CORSIKA File
  This is the output of CORSIKA. It contains simulated air showers propagated to the ground. The energetic particles that reach the ground are preserved in the file, but no detector simulation is included.
- HAWCSim File
  HAWCSim is our GEANT-4 application for simulating the ground array. The HAWCSim output file is an XCDF-formatted file that contains, for each PMT sensor in HAWC, the amount of light resulting from a simulated shower. Part of the PMT response is simulated at this stage and part of the response is simulated at the reconstruction stage. To maintain maximum flexibility, the full HAWC instrument is simulated at this point. The exact PMT/Tank layout is modeled at a later stage by masking out PMTs/Tanks that aren't deployed or operating.
- TRIG File
  This is an XCDF-formatted file produced by the DAQ/Online system that contains HAWC events. This file format contains the raw TDC edges grouped into 2- and 4-edge events with flags for various systems in the DAQ. This is "the data" that HAWC acquires and is the vast majority of the data volume that we preserve.
- REC File
  This is an XCDF-formatted file that contains tightly-compressed summary of the reconstruction procedure. The data in this format represent some 10% of the original TRIG data. All hit information is stripped out and not duplicated in these files. This data is produced both by the offline and online reconstruction procedures. As we expect a significant disk bandwidth bottleneck at the map-making stage subsequent to the generation of this data, we want it to be as compact as possible and an effort is made to keep unnecessary data out of this file format. As such, it can be difficult to use this format for in-depth study of HAWC events. If your favorite parameter wasn't computed already the reconstruction will have to be re-run to compute it.
- Full-Hits File
  This is a ROOT file that contains essentially all data generated in the reconstruction procedure. The number and types of the fields in this file are determined dynamically. For instance if one adds a new hit selection to the file, it is very easy to get that additional hit selection outputted to the file. When generated from simulated data, the ROOT file contains a branch specifically for the MC Truth. There are options in the programs that produce this data to suppress or add fields. This is the simplest, most friendly format for studying what is going

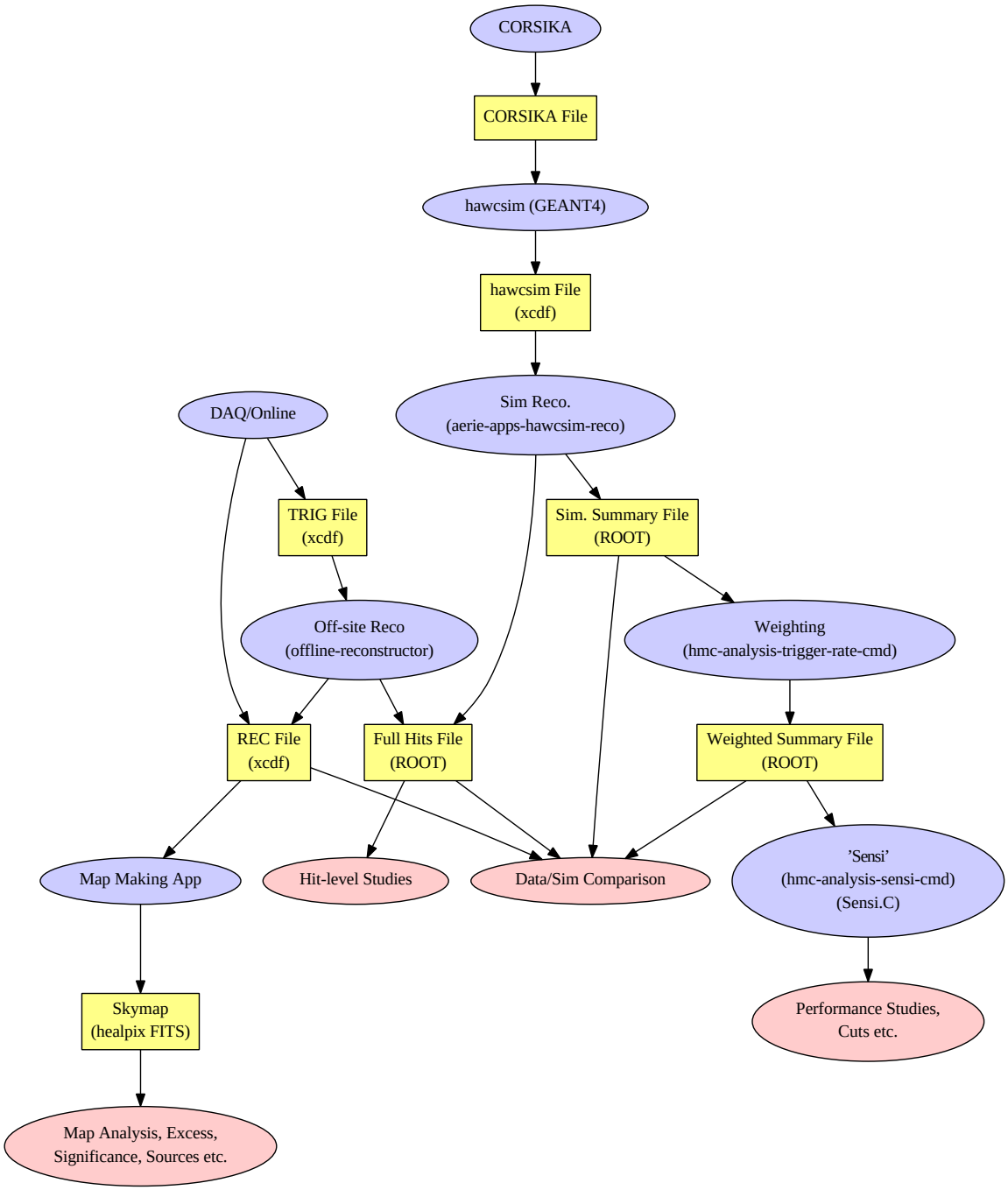

Figure 1: HAWC main triggered-data analysis formats and applications. Data is shown in yellow boxes, production applications in blue ovals, and analysis in pink ovals.

on in the reconstruction, but this comes at a cost. These files are large. It has a number of redundant branches. It is impractical to save this data for any significant number of events. Nevertheless, it is probably the most convenient source for "getting to know" the data. The MC version of these files carry a limitation that they do not contain weights and there is currently no code to compute the weights for these files. There are some tricks to get around this limitation that will be described later.

- Simulation Summary File
  This is the simulation analog of the REC data. It is a fairly compact summary of reconstructed events, saved as a ROOT TTree. It does not include "hit" information and usage of this file generally presumes that your favorite parameter is already included in the Trees. If it wasn't, most likely the reconstruction will have to be re-run. Generally, the simulation reconstruction is run and this output is saved for each input file. Then the files are combined into a single file of this type for each particle species, (I.e. gamma.root, proton.root, helium.root etc). These files are generally only useful as an intermediate stage to the production of the weighted files.
- Weighted Simulation Summary Files
  These files are largely identical to the Simulation Summary Files except they have to additional branches "Wgt" and "TWgt". These are the simulation weights. In the preparation of these files, a physical flux of cosmic rays is assume, as is an energy spectrum and declination of a gamma-ray source. We will discuss later how to use these weights to get predictions about the detector.
- Skymaps
  The final main supported data format, skymaps, are a bit beyond the scope of this document, but I include them so that the reader can see how they fit into the big picture. The skymaps we use are generated from the experimental REC data.

These file formats are largely the process of an organic growth from "what we did in Milagro." There are a number of stupidities in this list. For instance, there is no good reason that the REC data cannot more closely match the Weighted Simulation Summary Files. They contain largely the same data, but with different names for the branches and other irritating things. Also, there is no good reason that the REC data can't be simply a handful of branches selected from the full trees. As it is, a script or program that runs on the REC data must be modified to run on the "Full Output" data. Perhaps most vexing, it is currently not possible to provide absolute weights to the "Full Output" ROOT TTrees. The simulation weighting only works with the summary trees. Nevertheless, this is what we have at the moment.

Generally speaking, if you are just interested in playing around and learning you will want the "Full Output" ROOT files. They are a great way to get started understanding how the reconstruction works and what the data is. The REC data and the Weighted and Unweighted Simulation Summary files are useful when you are making maps of the sky or evaluating the sensitivity of HAWC.

It's also worth noting that there are other formats and data paths not included in this simple treatment. For instance, the sensitivity paper took the Weighted Simulation Summary Files as a starting place and went from there, having a number of intermediate processing steps and file formats.. Also, the generation of Data Challenge data has its own processing chain and intermediate file formats and isn't covered in this treatment. I also left off the raw data, tdcscaler data, and others because "most" users won't interact with them.

3 Working with Full ROOT Output

If you're just getting started, get one of these "Full ROOT Output" files and play with it. If you don't have one, get some triggered data, build aerie, and run the offline-reconstructor on it with the "−−r" option to generate this output.

The TTrees in the resulting files are big. Pretty much each processing step results in an object that gets inserted into the HAWCNest Bag. The code includes a way for many Bag objects to get dumped to ROOT or xcdf so, by and large, each branch represents an object in the Bag and a step in the processing. I'm not going to describe every leaf in the TTree, but here are some tips to get you started: The events.XXX branches represent the raw and calibrated basic hit information. A "hit" here is either a 2-edge or 4-edge event in a PMT, so there may be more hits than channels in the detector. The leaf event.nHit is important because it is the number of entries in all "hit" branches. Note that this is a pseudo-automatic dump of data in the bag, so the units may not be particularly friendly. Angles are given in radians, for instance.

In simulated output, the SimEvent.X branch is the branch that includes the MC Truth. You'll also have output from the two steps in the core reconstruction, the CoreGuess.XXX (formerly the Center-of-Mass or COM core) and CoreGauss.XXX, the Gaussian fit to the core. You'll have output from the two steps in the angle fitting as well, planeFit.XXX and lhFit.XXX. There will probably be a number of compactnessN.XXX branches for various-radii compactness calculations. Finally, a number of hit selections may be included. Typically stdCuts.XXX is the branch for the hit selection used in the reconstruction, and cxpeCuts.XXX is the somewhat-smaller selection used for computation of the compactness values. Note the leaf, for instance, stdCuts.isSelected. That is an array branch, like all the hits, and can be used to distinguish hits used in the reconstruction from ones that weren't. You can see this by plotting "event.hit.time" with and without "stdCuts.isSelected" in the cuts field.

Working with the simulated output carries a special warning. Recall the simulation has intentional biases. The code will currently not compute the Weights for the "full output" simulation TTrees. There's no deep reason, it just hasn't been done. If you want to get a realistic weighting you can use the weight "sqrt(pow(SimEvent.xCoreTrue,2) + pow(SimEvent.yCoreTrue,2)) * pow(SimEvent.energyTrue,-0.7)". That will, for cosmic rays, have a roughly right weight up to an overall scale factor.

4 Working with Weights - Isotropic Sources

The Weighted Simulation Summary files are particularly useful for generating expectations because of the inclusion of the weights. This section describes how to interpret these weights and use them in the context of analysis. I'm not describing the computation of these weights, just what they mean and how to use them. The computation of these weights is described elsewhere[1].

There are two situations we want to use these files to model. The first is to model the "isotropic" case, where we just gather data from all directions. This is typically the situation when we're studying the cosmic-ray background. The second is the "transiting source" case, where we want to model what we'll get, signal and background, from a source that crosses overhead. In this second case, we assume we have a small fixed-size angular bin and that we are following a source of a fixed Declination.

The Wgt field is used for weighting the "isotropic" case. Note that though one may talk about an isotropic gamma-ray population, the code does not handle this situation yet. Consequently the Wgt weight, used by itself, only makes sense for hadronic background.

The Wgt branch is computed so that, when used as the weight in a TTree::Draw() method, that the result will be our expectation for one second of data. Further cuts may be applied and,

as long as the Wgt branch is used as a weight, the resulting figures will represent an "un-doing" of all the simulation biases and a scaling to one second of data. Note that currently both photon simulation and hadron simulation reside in the same TTree, so it is necessary to cut on the particle ID, "PID¿1" to select only hadrons.

To see this, consider the following operation on one of the HAWC-111 Weighted Simulation Summary TTrees:

- HMCEvents->Draw("nHit>>histoNHit","(Wgt)*(nHit>18)*(PID>1)");
- histoNHit->Integral();
- (const Double_t)1.18195864133834839e+04

which is darned close to our trigger rate for HAWC-111, as it should be.

5 Working with Weights - Transiting Sources

The transiting case is a bit more complicated than the isotropic case because of the nuances of a point source analysis. When working with a transiting source, one uses the combined weight (Wgt)*(TWgt). Note that TWgt makes no sense on its own. Because, when following a bin through a transit, we will get both signal events and hadronic background events, the (Wgt)*(TWgt) weight applies equally well to both the photon and hadron cases. One can select whether they want the answer for photons or hadrons by cutting on "PID==1" (for photons) or "PID>1" (for hadrons).

The TWgt carries the part of the weight that accounts for the transit and is strongly dependent on the Declination of the source. It is basically a function of the Theta, accounting for how much time a source of the specified Declination spends at each Theta in the detector. Currently, the code that computes weights (hmc-analysis) takes as input the Declination and photon source spectrum that you want to model and these are "burned in" to the file in the Wgt and TWgt parameters. If you want to model another Declination source, hmc-analysis must be re-run. Because the natural unit of a transiting source is one day (one transit of the source) the (Wgt)*(TWgt) parameter is scaled so that it gives the answer for one transit of the source.

Another subtlety is the bin size used. The background expectation scales as the solid angle of the bin used, where the signal expectation scales according the point spread function. When hmc-analysis is run, a nominal bin sized is presumed. Typically, we assume 2.5 deg (much larger than any bin size we would want to use). If you want to actually model a source, then, you have to specify a smaller bin size. The background must be scaled appropriately and you have to cut on DelAngle for the photons, to mock up the cut on a bin around the source.

Let's walk through this with my HAWC-111 file. You'll probably get a slightly different answer, unless you use the exact file I'm using, when you try this but they should be pretty comparable.

Start by looking at the plot:

- HMCEvents->Draw("DelAngle","PID==1");

This distribution ends, for me, at 2.5 degrees, indicating that this file was, in fact, prepared with the standard 2.5 deg bin assumed. Now I want to figure out how many photons I'll get from the source assuming a more realistic bin size of 1 deg.

First, the signal expectation:

- HMCEvents->Draw("nHit","(PID==1)*(TWgt)*(Wgt)*(nHit>18)*(DelAngle<1.0)");
- htemp->Integral()
- (const Double_t)2.96588099621236324e+02

or about 300 events in a transit.

Now let's look at the computation of the background rate. This file was prepared assuming a 2.5 deg bin, but I want to look at 1 deg. That means after getting my answer from the ROOT TTree, I'm going need to scale by the ratio of the solid angles:

- $(1. - \cos(1.0^o))/(1 - \cos(2.5^o)) \approx 1.0^2/2.5^2 = 0.16$
- HMCEvents->Draw("nHit","(PID>1)*(TWgt)*(Wgt)*(nHit>18)");
- htemp->Integral() * 0.16;
- (const double)1.87263469377441419e+05

It is critically important to note the difference between the photon and background calculations. To mock up the bin we cut on DelAngle in the signal case. In the background case we do not cut on DelAngle (events are background if they end up in the bin, whether they were reconstructed accurately or not) and instead we scale from the pre-computed 2.5° case to the angular bin size we want to use.

So, in one transit of the Crab, with no cuts, in a 1° circle around the source, I expect 300 events from photons and 1.8×10^5 events from background. Not great, but we haven't applied any cuts yet, nor used the fact that we expect tighter angular resolution at high nChannel.

The Sensi programs that people are using to compute cuts basically make this operation over and over and determine what cuts will maximize the statistical significance of a Crablike source. The linchpin of these calculations is the weights we just looked at. That's what makes the Sensi calculation tick.

Finally, there is the issue of Gaussian Weighting. In the Gaussian Weighting situation, the calculation is a little more involved. See more recent versions of Sensi. But it all stems from the use of the Wgt and TWgt weights appropriately.

6 XCDF File Format

XCDF is the file format we use for most of our mission-critical data. It is a format dedicated to the notion that no bit should go unused. When declaring fields, the user specifies the precision to which they wish to save a result and the remaining precision is truncated. Aside from this feature, an XCDF file is a lot like a ROOT TTree and you should think about them similarly.

Working with an XCDF file directly is a lot like you'd expect. You're best off to find an example and copy it. I'm not going to discuss that because I figure if you're doing that you're gonna be able to figure it out pretty easily and XCDF is documented elsewhere[2].

Instead, I want to point out a couple really useful command-line utilities that make working with XCDF very easy. Again, these are documented elsewhere, I just want to point them out.

xcdf-utility is a great little utility for digging into an XCDF file. It has a help, so just run it without arguments to see the help. Check out, particularly:

- xcdf-utility count
  which will tell you how many events are in a file.
- xcdf-utility info
  which will tell you what the fields are in a file.
- xcdf-utility select
  which will generate a new XCDF file based on a down-selection of parameters. I use this utility to select REC data that reconstructed near the Crab, for instance. We also use it to generate the high nChannel triggered dataset.

- xcdf-utility dump
  which will make an ascii dump of a file.

and so on. The real strength, though, is that the utilities are designed as standard unix utilities which can be chained together with pipes. So a command like

- xcdf-utility select "rec.nChannels >= 250" | xcdf-utility count

will tell you how many events in the file have at least 250 hit channels.

I also personally make heavy use of a utility 'xcdf-root' which will convert an xcdf file into a ROOT file with a TTree named XCDF. It's useful for exploring what's in a file you have.

Finally, a new utility in aerie allows the selection of triggered data based on some conditions in the reconstructed data is very useful for selecting triggered events for futher study. The aerie-apps-xcdf-time-selection program accepts a triggered file as an argument and a comma-separated list of GPS times on stdin. It can be chained with the xcdf utilities as follows:

- xcdf-utility select "((rec.ra - 1.434) < 0.0045) && (1.434 - rec.ra) < 0.0045) && (((rec.dec-0.349) < 0.0045) && ((0.349 - rec.dec) < 0.0045))" REC.xcd | xcdf-utility select-fields "rec.gpsSec,rec.gpsNanosec" | xcdf-utility csv | aerie-apps-xcdf-time-selection TRIG.xcd > OUT.xcd

That, admittedly arcane, command line will select all events that reconstructed near the Crab in REC.xcd, find their GPS times, and then down-select the triggered data in TRIG.xcd, finding the triggered events that these reconstructed events come from.

7 Adding Fields to Full ROOT Output

We have a pretty sophisticated suite of templates (written by Jim Braun) that handle writing HAWCNest Bags to file. The main purpose is to allow both ROOT and XCDF formats. XCDF is unique among formats in allowing the user to specify the precision to which they want to save results. Furthermore, the code is dynamically extensible, meaning that we we add types to write to file we need only write a fairly tiny class describing our data. This is all wonderful to somebody who just loves code like this (like me) but to a more typical user the reaction may be "Where the hell is the code that makes the ROOT file?!" Stop looking. You'll never find it.

In lieu of a poem of the beauty of templates – no doubt it would be a P about the B of T, only specifying that P is Poem, T is CxxTemplates and B is Beauty at the very end, so that the user can also choose to have a Rant about the Ugliness of a Hack instead – I'll give you a little boilerplate code to copy and modify. See Example 1. The idea is that you're hacking on the reconstruction program (either the one that reconstructs simulation, or the one that reconstructs data). This code can go right in the main() right before main starts. You'll have to add the PretzModule to the chain of modules. You'll also have to append "pretz" to the vector of results that is used to configure the DynamicSerializer object.

A Summary of Data Format

- Corsika Output File
  Purpose: Store the ground particle content of thrown primary particles.
  Format: Corsika file format defined in the Corsika library.
  Location: $HAWCROOT/sim/corsika
  Documentation: Monte Carlo Products, the most recent productions.

```
struct PretzParameter : public Baggable{
  double p2k;  // nonsense
  double pErr; // almost always 0 ;P
};

#include <aerie-io/serializer/baggable/BaggableSerializerCRTP.h>
#include <aerie-io/serializer/baggable/BaggableSerializerRegistry.h>
#include <aerie-io/serializer/baggable/SerializerConfiguration.h>

// structure that does the magic that turns your class into ROOT Tree or
// XCDF. Pretty much just follow this boilerplate code.
struct PretzSerializer :
  public BaggableSerializerCRTP<const PretzParameter, PretzSerializer>{
  void DoSerialization(Writer& w, const std::string& name,
                       const PretzParameter& p,BagPtr bag,
                       const SerializerConfiguration& configuration) const{
    w << FPWC(name + ".p2k", 0.001) << p.p2k;
    w << FPWC(name + ".pErr", 0.001) << p.pErr;
  }
};

// Dynamically registers the serializer so that the serialization framework
// knows about it.
REGISTER_SERIALIZER(PretzSerializer);

// Module to sit in the processing and compute your variable
class PretzModule : public Module{
public:
  typedef Module Interface;

  Module::Result Process(BagPtr bag){
    boost::shared_ptr<PretzParameter> p(new PretzParameter());

    // in reality, this would be computed
    // from data in the event
    p->p2k = gRandom->Gaus(0,1);
    p->pErr = 0.0;
    bag->Put("pretz",p);

    return Module::Continue;
  }

};
```

Example 1: Adding parameters to the ROOT file

- hawcsim Output File
  Purpose: Store photoelectrons resulting from air shower on the ground.
  Format: We use Jim Braun's XCDF format for these files. You will find other files around in a gzipped ASCII format. Unless there's a good reason, the XCDF should be used.
  Location: $HAWCROOT/sim/hawcsim
  Documentation: Monte Carlo Products, the most recent productions.
- Simulated Summary ROOT File
  Purpose: Store summary information from event simulation. Very concise, storing only a few reconstruction summary parameters from each event.
  Format: ROOT TTree with ancillary numbers stored in some TNamed elements in the file. One file for each of the 9 species simulated, namely gamma.root, proton.root, helium.root ... iron.root.
  Location: $HAWCROOT/sim/reco/*/*/combined
  Documentation: Monte Carlo Products, the most recent productions.
- Weighted ROOT File
  Purpose: Store individual event information from simulated events. Includes weights to mock up physical fluxes. Note that a photon source spectrum and declination are selected when making this file so there is at least one file for each declination, depending on how fine we want to be.
  Format: ROOT TTree with anciliary numbers stored in some TNamed elements in the file.
  Location: $HAWCROOT/sim/reco/*/*/hawc-srcs/*.root (easy to generate yourself too)
  Documentation: Monte Carlo Products, the most recent productions.
- Full-Hits ROOT File
  Purpose: Store information from event reconstruction. Not concise at all. Stores every hit and nearly every step in the processing.
  Format: ROOT TTree with anciliary numbers stored in some TNamed elements in the file.
  Location: $HAWCROOT/sim/reco/*/*/[gamma, proton,... iron]/suceeded/*.root (not mmcevents files) We will probably reconstruct some subset of the experimental data this way but it's not done regularly yet.
  Documentation: Monte Carlo Products, the most recent productions.
- TRIG File
  Purpose: Store real-data triggered events. Stores edge-found ToT values for each PMT in an event.
  Format: XCDF file.
  Location: $HAWCROOT/data/hawc/data/YYYY/MM/runXXXXX/trig_*.xcd
  Documentation: http://private.hawc-observatory.org/svn/hawc/documentation/data-format with thanks to Colas for corralling that together.
- REC File
  Purpose: Store reconstruction summaries.
  Format: Internal DAQ format, wrapping data taken straight from the TDCs.
  Location: $HAWCROOT/data/hawc/data/YYYY/MM/runXXXXX/reco_*.xcd (on-site reconstruction) $HAWCROOT/data/hawc/reconstructed (off-site reconstruction)
  Documentation: http://private.hawc-observatory.org/svn/hawc/documentation/data-format again, thanks to Colas.
- Skymap File
  Purpose: Store measurements and determined backgrounds on the sky.
  Format: FITs file storing a healpix-interpreted table.

Location: Where: No standard location yet.
Documentation:

B References

1. Pretz, J. Simulation Weighting.
   http://private.hawc-observatory.org/hawc.umd.edu/internal/doc.php?id=2235 April, 2013.
2. Braun, J. XCDF Github Page.
   https://github.com/jimbraun/XCDF