Reference Manual

Generated by Doxygen 1.8.5

Fri Oct 7 2016 13:20:04

Contents

Chapter 1

GiBUUTools

GiBUUToStdHep is a tool for converting the default GiBUU, text-based neutrino-mode event output into a format that is more commonly used in the neutrino generator community. It aims to preserve as much of the useful information provided by the generator for subsequent analysis.

GiBUUFluxTools is a tool for converting neutrino flux histograms from bin-edge text histograms or root files containing a TH1 to a bin-center text histogram that GiBUU can use to throw neutrino events.

Building GiBUUTools

To build GiBUUTools:

- Make a build directory: `mkdir build`
- Configure the build: `cd build && cmake ../`
- Optional – If you want to download, patch, and build a local version of GiBUU2016 use `cmake /path/to/source -DUSE_GIBUU=1` instead.
- Build! `make`.
- Optional: Build the documentation – `make docs`.
  - This release should come with pre-compiled documentation at `dox/GiBUUTools.pdf`
- Install: `make install`.

To set up a built GiBUUTools:

- `source build/Linux/setup.sh`
  - `Linux` will change depending on your `CMAKE_SYSTEM_NAME`

This will add GiBUUTools and GiBUU (if built) to the PATH.

IMPORTANT

Whenever you plot properties of GiBUU events, you **must** weight the events by the 'event weight' (`EvtWght` in the stdhep tree). GiBUU throws events according to the neutrino flux shape, if you do not weight by the xsec, then your distributions *will* be just **wrong**.

Chapter 2

My First GiBUU Prediction

Here should be the fastest way to make a GiBUU prediction.

Build

- Make a build directory: `mkdir build && cd build`
- Configure the build: `cmake ../ -DUSE_GIBUU=1`
- Build! `make`.
- Install: `make install`.

Source the environment:

- `source build/Linux/setup.sh`
  - `Linux` will change depending on your `CMAKE_SYSTEM_NAME`

Generate some events

- `$ Generate_MiniBooNE_numuCC_CH2_Events_GIBUU`
- Make a coffee, or three.

If you want to up the stats, pass an integer as an argument to the `Generate_MiniBooNE_numuCC_CH2_-Events_GIBUU`, this will set `input:num_runs_SameEnergy`. This will increase the time to generate linearly with the number passed. Passing `10` will generate O(1.6E5) events and take about an hour on any semi-current CPU.

Plot the muon momentum

```
root -l MiniBooNE_CH2_numuCC.stdhep.root
[root] giRooTracker->SetAlias("DeltaE","StdHepP4[0][3]-StdHepP4[2][3]");
[root] giRooTracker->SetAlias("Delta3Mom0","StdHepP4[0][0]-StdHepP4[2][0]");
[root] giRooTracker->SetAlias("Delta3Mom1","StdHepP4[0][1]-StdHepP4[2][1]");
[root] giRooTracker->SetAlias("Delta3Mom2","StdHepP4[0][2]-StdHepP4[2][2]");
[root] giRooTracker->SetAlias("nQ2","(Delta3Mom0*Delta3Mom0 + Delta3Mom1*Delta3Mom1 + Delta3Mom2*Delta3Mom2) -
[root] TH1 *NQ2 = new TH1D("NQ2",";Q^{2} (GeV^{2});d#sigma/dQ^{2} (cm^{2} GeV^{-2})",20,0,2);
[root] NQ2->Sumw2();
// Weight histogram fills by the event weight and select only CCQE events.
[root] giRooTracker->Draw("nQ2 >> NQ2","EvtWght*(GiBUU2NeutCode==1)");
// Scale to a differential xsec per neutron
[root] NQ2->Scale(14.E-38/6.,"width");
```

```
[root] NQ2->Draw("E1");
// Compare to data
[root] TGraph comp("MiniBooNE_1DQ2_numu_CCQE.dat");
[root] comp.Draw("L SAME");
```

This is contained within a `cint` macro and can be executed as `root -l MiniBooNE_CH2_numuCC.-stdhep.root ${GIBUUTOOLSROOT}/cint_macros/Plot_MiniBooNE_CH2_CCQE_Q2.C`

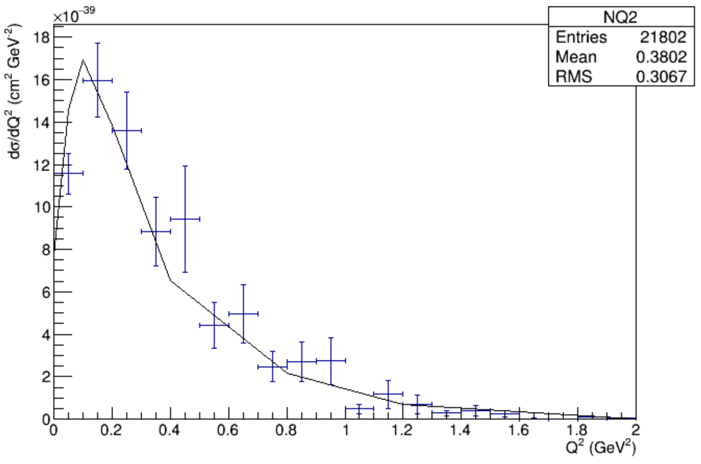

Figure 2.1: Example comparison of generated GiBUU CCQE events to the MiniBooNE data: Solid line is interpolated data, points are the generated prediction

If the normalisation looks similar to the above plot, then you're doing something right, if it looks awful, or you can't see both series, then something has gone wrong somewhere. Please report this as an issue on the GiBUUTools repo!

For all your comparison needs, consider using `NUISANCE`, a generic global xsec generator tuner and comparison synthesiser.

Look at other fluxes/target

Other helper scripts that are provided:

- `Generate_ND280_numuCC_CH_Events_GIBUU`
- `Generate_MINERvA_numuCC_CH_Events_GIBUU`
- `Generate_DUNE_numuCC_Ar_Events_GIBUU`
- `Generate_NuMi_numuCC_Ar_Events_GIBUU`

To generate a serious amounts of events, using a batch system is advised. **Important:** Remeber to change the `SEED` for each job!

Chapter 3

Running GiBUU

Having sourced the installed setup script (`/<install path>/setup.sh`), check that you can see GiBUU by executing: `which GiBUU.x`. If you can't, you probably haven't built, configured with `-DUSE_GIBUU=1`, or sourced the setup script correctly.

Some jobcard options are explained below. To run a simulation from a given jobcard, execute: `GiBUU.x < my.job`. This will output O(100) files, so it is probably worthwhile running it in a subdirectory of anywhere you care about. The main file that we will use is `FinalEvents.dat`, which is the text event vector. If this was not produced when running, check that `neutrinoAnalysis:outputEvents=.true.` is set.

Useful jobcard options

Options specified throughout take the form: `NamelistName:ParameterName = ParameterValue`. In the jobcard these will look like:

```
&input
  numEnsembles=4000 ! Number of nuclei to simultaneously model per run.
  eventtype=5       ! Neutrino induced event
  numTimeSteps=150  ! Number of hadronic transport steps to simulation
  !...
/
```

where "'input'" is the `NamelistName`, and `numEnsembles` is a parameter within that namelist.

Note: You must set `input:buuinput` in each jobcard. You can make a symlink in the current working directory to the downloaded inputs folder by executing `MakeBUUInputSoftlink`.

Note: The character string that `input:buuinput` is read into is quite limited in length (O(100) characters), if the local path is long, consider using a symlink to make shorter, or edit the code to make the variable longer. It can comprehend relative paths so 'input:buuinput='./BUUInput'' is an acceptable value.

GiBUU was not designed as a neutrino generator and thus has a significant number of options that you are unlikely to want to play with from a neutrino interaction physics standpoint. A good base jobcard can be dumped to the current directory by using the command `GiBUUInitNeutrinoJobCard`. More default jobcards can be found in `<install path>/jobcards`.

Interaction physics options:

Target nuclei and Nucleon momentum distributions

The nuclear momentum distribution model used is a local Thomas-Fermi gas and it's implementation is described in Phys. Rev. C 79 034601 Section III.A. For some nuclei, explicit density parameterisation and experimental tunings from Nuclear Physics A 554 4 509 (1993) and Atomic Data and Nuclear Data Tables 14 5 479 (1974) can be used (`target:densitySwitch_Static=2`). However, for some nuclei the specific experimental data is not available within GiBUU, and thus `target:densitySwitch_Static=3` should be used — notably to simulate on Ar40 you must use the Wood-Saxon density.

```
&target ! e.g. A Carbon-target set up
    target_Z=6
    target_A=12
    densitySwitch_Static=2 ! 0: density=0.0
                           ! 1: Wood-Saxon by Lenske
                           ! 2 : NPA 554
                           ! 3: Wood-Saxon by Lenske, different neutron
                           !    and proton radii
                           ! 5: density distribution is a sphere with
                           !    density according to the input value of
                           !    "fermiMomentum_input".
    fermiMomentum_input=0.225 ! Only relevant for "densitySwitch_Static=5"
    fermiMotion=.false.
    ReAdjustForConstBinding=.false.
/
```

For much much more detail see here, references therein and here.

QE

Quasi-Elastic interactions are modelled similarly to other generators, and by default use form factors from BBBA07 and an MA of 1.03.

```
&ff_QE
  parametrization=3         ! 1=BBBA03, 2=BBBA05, 3=BBBA07
  useNonStandardMA=.false. ! if true, use value of MA_in for axial mass MA,
                           ! if false, use best fit
  ! MA_in=1.03
/
```

The implementation is described in detail in Phys. Rev. C 73 065502 (2006).

2p2h

The 2p2h model used in GiBUU2016 is the Christy model () and is tuned only to electron scattering data. The implementation is described in detail in Phys. Rev. C 94 035502 and is accompanied by numerous model comparisons to recent experimental data.

```
&lepton2p2h
  ME_Version=4 ! This is the Christy model
  T=1
/
```

Note, in (Phys. Rev. C 94 035502), `T=1` (target isospin) is used and it is likely that it should be used for your comparisons.

Pion production

Model: Nuclear resonance-induced single pion production with tuned, non-resonant 1- and 2-pi background contributions. By default, the resonance matrix elements are calculated from the MAID model (Eur. Phys. J. A 4 1 69 (2007)). It is also possible to use Rein-Sehgal, or just a simple Delta resonance.

The overall parameter for controlling the resonant pion production model is `neutrino_matrixelement::which_resonanceModel`. The options are shown below.

```
&neutrino_matrixelement
  which_resonanceModel=0     !0=QE + matrixelements from MAID,
                             !1=QE matrixelements + old Delta,
                             !2=Rein-Sehgal
/
```

The model has been tuned to both ANL and BNL bubble chamber data. Both the resonant and non-resonant contributions should be altered depending on which tuning is in use.

For ANL fit:

```
  &input_FF_ResProd
    FF_ResProd=0  ! 0=MAID in CM-frame
    MA=0.95       ! Axial mass in the Delta resonance form factor
  /
  &neutrino_MAIDlikeBG ! Non-resonant background contribution
    b_proton_pinull=3.0
    b_neutron_piplus=1.5
  /
```

For BNL fit:

```
&input_FF_ResProd
  FF_ResProd=0 ! 0=MAID in CM-frame
  MA=1.3       ! Axial mass in the Delta resonance form factor
/
&neutrino_MAIDlikeBG ! Non-resonant background contribution
  b_proton_pinull=6.0
  b_neutron_piplus=3.0
/
```

Extensive comparisons to MiniBooNE data can be found in `Phys. Rev. C 87 014602`. The systematic uncertainty on the GiBUU model is generally taken as the difference between predictions using the ANL and BNL fit, as in (Phys. Rev. C 87 014602).

The implementation is discussed in `Phys. Rev. D 82 093001`.

Got your own flux?

GiBUU includes quite a number of pre-defined fluxes that can be enabled with the `neutrino_induced:nuExp` entry in the jobcard. The default jobcard is well documented with regard to which values correspond to which flux.

However, sometimes you might want to use your own flux. By default, GiBUU does not expose simple options for reading arbitrary fluxes. The version that can be built here (using cmake option `-DUSE_GIBUU=1`) is patched so that arbitrary fluxes can be specified in the jobcard. To do so you will want to set `neutrino_induced:nuExp = -1` and then ‘neutrino_induced:InputFluxFileName='/path/to/flux/file.txt'‘.

Note: The character string that this is read into is quite limited in length (O(100) characters), if the local path is longer, consider using a symlink to make shorter, or edit the code to make the variable longer.

Note: So as not to collider with the internal usage of the flux reading method you must specify the value of `neutrino_induced:InputFluxFileName` as either a fully qualified path, or a relative path beginning with `./`.

The fluxes are simple two column, space delimited tables of flux shape. Note that the normalisation is entirely ignored. The first column is taken as bin centre values, and the second as the flux content in that bin. Consider using the packaged `GiBUUFluxTools` helper to convert from bin-edge text histograms or ROOT files containing histograms to GiBUU friendly flux inputs.

Want to run without FSI

The simplest way to 'run without FSI' is to set the number of hadronic transport steps to 0 (`input:numTime-Steps=0`). This will result in an 'inclusive' analysis of the uncoupled neutrino interaction model. However, as the final state particles will not have been transported through the nuclear medium they will still feel the potential of of the nucleus and their kinematics will not be free. When examining the kinematics of such particles, be aware that they are likely still off-mass-shell. This can still be a useful way to produce plots with that caveat.

Want to run on free nucleons?

To generate events on hydrogen, either for comparisons to bubble chamber data, or for building composite targets such as CH, the following options should be used:

```
&input
      numTimeSteps=0
/
&target
      target_Z=1
      target_A=1
      densitySwitch_Static=2
      fermiMotion=.false.
      ReAdjustForConstBinding=.false.
/
&width_Baryon
    mediumSwitch=.false.  ! if .false. vacuum widths will be used for all
                          ! resonances
/
```

As the majority of the time-consuming calculation is running the hadronic transport simulation, this will execute very quickly.

How many events to generate

Unlike for other generators, which do not have coupled initial and final states, before the simulation is run it is not possible to exactly determine all allowed phase spaces. This means that if you want 1E6 events, it is impossible to ask the generator to generate exactly this many events. The upper-limit of the number of events generated is given by: `target_A * numEnsembles * num_runs_SameEnergy`. To determine the number of events you should execute a short run with the proposed numbers for `target_A` and `numEnsembles`... and then scale up statistics by increasing the value `num_runs_SameEnergy`. This should give a predicatble number of events (though still not exact).

Modifying `input:numEnsembles` sets the number of nuclei to simultaneously simulate. Increasing it, improves the efficiency of the simulation, but will increase the memory usage. `input:numEnsembles=4000` is a good start for a carbon target, but can be decreased for heavier nuclei.

How long to run the simulation for

The number of simulation steps is set by the jobcard. If particles are still inside the nucleus at the end of the simulation, their kinematics will show them as off-mass-shell. A good starting point for a carbon-target is to set `input:numTimeSteps=150`, larger nuclear targets will require more simulation steps. The suggested value is such that `input:numTimeSteps*input:delta_T (fm)` should significantly exceed the radius of the target nucleus. You can also check the number of particles still inside of the nucleus at the end of the simulation by looking at the output.

```
################### NEUTRINO ANALYSIS FINISHED #######################
(FINAL) number of counted Events:
Real         -- all, 1-body, 2-body, 3-body:        0        0        0        0
Perturbative -- all, 1-body, 2-body, 3-body:    30529     4890    24867      772
```

The numbers shown here are the number of interactions still taking place at the end of the simulation. These numbers should decrease when increasing `input:numTimeSteps`, for a given targer. Ideally these would all be 0.

Output histograms

It appears that often the GiBUU developers code their analyses into the simulation itself, as a result many in-built analyses can be run at the same time as generating the events. For a number of uses, these results are actually all the information you might need. The options are contained within the namelists `neutrinoAnalysis` and `nl_specificEvent`. Turning some of these analyses on will significantly increase the number of output files that GiBUU produces.

FAQs

Why are lots of my final state particles off shell? GiBUU will transport the final states of any neutrino interactions through the hadronic transport simulation for an exact number of steps. If some of these particles, after the specified number of steps, are still inside the nucleus, they will feel the nuclear potential and thus be off-mass-shell. By plotting the calculated mass of certain classes of final state particles you should be able to gauge how much of a problem this is for you events. If it appears to be a significant problem, then you should up the value of `input:numTimeSteps`. O(150) appears to be okay for Carbon-targets.

Why does GiBUU fail to start when given a flux file with more than 120 bins? Because FORTRAN. The flux histogram is read into a compile-time allocated array. You have three choices: 1 – Chop off a few bins from the tail of your flux, 2 – Rebin your flux, 3 – Edit the size of the array in GiBUU and recompile.

Where do I go for more info?

The full GiBUU documentation is on hepforge: `GiBUU hepforge`. The GiBUU developer mailing list will often offer specifc help – they want their wonderful code to be used for good physics! Please do not spam this address before searching for the answer in papers and documentation.

Chapter 4

GiBUUToStdHep Command Line Interface

The options that you need to pass to correctly parse and combine the GiBUU output can be a bit fiddly at times. As GiBUU cannot generate events on composite targets, or by multiple neutrino species, or both CC and NC events in the same run, multiple, separate GiBUU runs must be consistently combined.

Note event when consistently combined, the raw events will not be correctly distributed, however, the event weights will be correct. **You must always weight event properties by the `EvtWght` when building histograms**.

Note: It is almost always easiest to increase the statistics of a single run mode by running multiple identical jobs. Correctly merging the cross section predictions from two simulations, run on the same target with the same neutrino species and CC/NC mode, but with differing `input:numEnsembles` or `input:numTimeSteps` is infeasibly fiddly. If you want to generate a large number of events, spread multiple identical jobs over multiple CPUs. **IMPORTANT**: Remember to change `initRandom:SEED` for each run, or your separate runs will be *too* identical.

Command line options are split into two types, ones that affect the whole parsing, and ones that affect only the next file or group of files.

Whole parsing options

- `(-s|--long-form) <Option Type {default value}> [Required or not]`: Example of an option described by this documentation.
- `(-h|--help)`: Print a help message with a usage example.
- `(-c|--CompositeExample)`: Print the example usage for building a full CH-target vector including neutrino and anti-neutrino flux components
- `(-o|--output-file) <FileName {default:GiBUURooTracker.root}>`: The output file name.
- `(-R|--Total-ReWeight) [i]<[1.0/]float>`: The overall weight to apply to the output vector, useful for outputting a composite-target vector with xsec weights in units of `/nucleon`. If the value is prepended with an `i` then the inverse of the numerical part of the option is used, e.g. if `-T i12` is passed, then the file weight will be `1/12`.
- `(-NI|--No-Initial-State)`: If you are using an old version of GiBUU which does not output initial state/target nucleon information this will not look for it. GiBUU2016 has initial state information in the output FinalEvents.dat
- `(-NP|--No-Prod-Charge)`: If you are using a default version of GiBUU, as opposed to the patched version that can be built by this package, if enabled, this will not expect that information. This makes guessing the NEUT-equivalent mode more tricky as you do not know the charge of the neutrino-induced resonance state.
- `(-v|--Verbosity) <0-4>`: Raises the verbosity of the parsing.

Options which affect the next input file(s)

As making a complete event vector with GiBUU always requires multiple runs, a number of CLI options will have to be specified for each file. e.g. Whether the next file contains NC or CC events, whether the next file contains Carbon-target, or Hydrogen-target. The options will be described before a few examples are given, which should hopefully make their use more clear.

- `(-u|--nu-pdg) <int> [required at least once]`: Specifies the neutrino species PDG of the next file(s). **Note:** This option is assumed for subsequent `-f` options until overriden.
- `(-N|--is-NC)`: Specifies whether the next file(s) are simulated NC events. **Note:** This option is assumed for subsequent `-f` options until overriden.
- `(-a|--target-a) <int> [required at least once]`: Specifies the nucleon number of the target used in the next file(s). **Note:** This option is assumed for subsequent `-f` options until overriden.
- `(-z|--target-z) <int> [required at least once]`: Specifies the nucleon number of the target used in the next file(s). **Note:** This option is assumed for subsequent `-f` options until overriden.
- `(-W|--file-weight) [i]<[1.0/]float>`: Specifies the overall file target weight for the next file(s). If the value is prepended with an `i` then the inverse of the numerical part of the option is used, e.g. if `-T i12` is passed, then the file weight will be `1/12`. **Note:** This option is reset to `1.0` for subsequent `-f` options, the next file(s) weight *must* be specified for each set of files to be parsed.
- `(-f|--FEinput-file) <File Name> [required at least once]`: Specifies the next file(s) to parse, which all previous 'per file' options will apply to. Wildcards are allowed at the file level of the specifier, but not at a directory level: e.g. `-f "some/subdir/FinalEvents*.dat"` is allowed but `-f "some/sub*dir/FinalEvents.dat"` is not. Averaging over multiple runs is handled automatically, so the file weight specified by `-W` does not need to account for multiple files being parsed due to the wildcard expansion. **Note:** Be careful not to let the calling shell expand the wildcard, when using a wildcard in the file specifier, wrap the path in double quotes, e.g.: `-f "path/to/some/files*.dat"`.
- `(-F|--Save-Flux-File) <output_hist_name,input_text_flux_file.txt>`: This option is used to save the GiBUU-style bin-centered flux histogram stored in `'input_text_flux_file.txt'` as a ROOT `TH1` named `'output_hist_name'` in the output file. This can be useful for some downstream code.

Notes on event weight combinations

As part of the generation, GiBUU averages cross section weights over the ensemble of generated targets (`input:numEnsembles`). *You never need to account for this averaging.* The averaging over weights for values of `input:num_runs_SameEnergy` larger than `1` is handled automatically by `GiBUUToStdHep` upon parsing each `FinalEvents.dat` file. *You never need to account for this averaging.*

You do need to account for averaging over the number of nucleons of a composite target. The cross section weights in the final event vector are often expected to be in units of per nucleon. While it might be expected that this weight handling could be taken care of by using the `-a` option, for clarity when building vectors of targets such as CH2 it was decided that this weighting should be explicit. Therefore, when combining runs for a composite target, the per nucleon weights needs to be manually applied: e.g. for a CH2 target, the overall output weight should be `-R i14`, the input carbon-target run weights should be `-W 12` and the input hydrogen-target run weights should be `-W 2`. This weights each contribution up correctly for the target that was simuatled in a given run (C or H), and then weights the combined results back down to per composite-target-nucleon. The resulting weights in the event vector should correctly give cross sections in units of `10^-38 cm^2 /nucleon`.

Composite target examples

To build an event vector of correctly weighted CH2 muon neutrino mode events where the Carbon-target events are stored in some subdirectory `numu_C_CC`, and the hydrogen-target in `numu_H_CC`, in files named `Final-Events_<RunNum>.dat` the command might be:

```
  GiBUUToStdHep\
    -u 14 -a 12 -z 6 -W 12 -f "numu_C_CC/FinalEvents_*.dat" \
      -a 1 -z 1 -W 2 -f "numu_H_CC/FinalEvents_*.dat"\
    -R i14 -o numu_CH2_CC.stdhep.root
```

Composite beam examples

To build an event vector that from multiple GiBUU runs that used different neutrino components of a realistic flux, where the Carbon-target anti-nue mode events are stored in some subdirectory `FHC_nuebar_C_CC` (FHC stands for Forward Horn Current, often synonymous with a 'mostly muon neutrino beam'), and other components follow a similar naming convention, the command might be:

```
GiBUUToStdHep \
  -u 14 -a 12 -z 6 -W 12 -f "FHC_numu_C_CC/FinalEvents_*.dat" \
    -a 1 -z 1 -W 2 -f "FHC_numu_H_CC/FinalEvents_*.dat" \
  -u -14 -a 12 -z 6 -W 12 -f "FHC_numubar_C_CC/FinalEvents_*.dat" \
    -a 1 -z 1 -W 2 -f "FHC_numubar_H_CC/FinalEvents_*.dat"\
  -u 12 -a 12 -z 6 -W 12 -f "FHC_nue_C_CC/FinalEvents_*.dat" \
    -a 1 -z 1 -W 2 -f "FHC_nue_H_CC/FinalEvents_*.dat"\
  -u -12 -a 12 -z 6 -W 12 -f "FHC_nuebar_C_CC/FinalEvents_*.dat" \
    -a 1 -z 1 -W 2 -f "FHC_nuebar_H_CC/FinalEvents_*.dat"\
  -R i14 -o numu_CH2_CC.stdhep.root
```

You can see that it is starting to build up... (In future there might be a card-file style option specifier).

Full example

And now to add in some NC events...

```
  GiBUUToStdHep \
    -u 14 -a 12 -z 6 -W 12 -f "FHC_numu_C_CC/FinalEvents_*.dat" \
      -a 1 -z 1 -W 2 -f "FHC_numu_H_CC/FinalEvents_*.dat" \
    -u -14 -a 12 -z 6 -W 12 -f "FHC_numubar_C_CC/FinalEvents_*.dat" \
      -a 1 -z 1 -W 2 -f "FHC_numubar_H_CC/FinalEvents_*.dat"\
    -u 12 -a 12 -z 6 -W 12 -f "FHC_nue_C_CC/FinalEvents_*.dat" \
      -a 1 -z 1 -W 2 -f "FHC_nue_H_CC/FinalEvents_*.dat"\
    -u -12 -a 12 -z 6 -W 12 -f "FHC_nuebar_C_CC/FinalEvents_*.dat" \
      -a 1 -z 1 -W 2 -f "FHC_nuebar_H_CC/FinalEvents_*.dat"\
  -N \
    -u 14 -a 12 -z 6 -W 12 -f "FHC_numu_C_NC/FinalEvents_*.dat" \
      -a 1 -z 1 -W 2 -f "FHC_numu_H_NC/FinalEvents_*.dat" \
    -u -14 -a 12 -z 6 -W 12 -f "FHC_numubar_C_NC/FinalEvents_*.dat" \
      -a 1 -z 1 -W 2 -f "FHC_numubar_H_NC/FinalEvents_*.dat"\
    -u 12 -a 12 -z 6 -W 12 -f "FHC_nue_C_NC/FinalEvents_*.dat" \
      -a 1 -z 1 -W 2 -f "FHC_nue_H_NC/FinalEvents_*.dat"\
    -u -12 -a 12 -z 6 -W 12 -f "FHC_nuebar_C_NC/FinalEvents_*.dat" \
      -a 1 -z 1 -W 2 -f "FHC_nuebar_H_NC/FinalEvents_*.dat" \
    -R i14 -o numu_CH2_CC.stdhep.root
```

For many applications this full example would be entirely uneccessary.

GiBUUFluxTools Command Line Interface

This tool prepares input flux files in either text or ROOT format for use by GiBUU. GiBUU expects to read two column text histograms laid out as: `<bin center> <bin content>`.

- `(-h|--help)`
- `(-l|--low-bin-edge) <int> [required for -t]`: Low bin edge column number, zero indexed.
- `(-u|--upper-bin-edge) <int> [optional for -t]`: Upper bin edge column number, zero indexed.

- `(-v|--value) <int>`: Value column number, zero indexed.
- `(-t|--input-file-text) <file path>`: Input text file name.
- `(-r|--input-file-ROOT) <file path>`: Input ROOT file name.
- `(-H|--input-ROOT-histogram) <string> [required for -r]`: Input ROOT histogram name.
- `(-o|--output-file) <file path> [required]`: Output file name.
- `(-k|--keep-norm)`: Whether to keep the input normalisation, GiBUU ignores the normalisation but can be useful to remember the normalisation.

When reading from an input bin-edge defined text histogram, `-l` must be specified, however `-u` is optional as the upper edge will default to the low edge of the following bin. For the upper edge of the final bin, if `-u` is unspecified, the bin width is assumed to be the same as the previous bin.

Chapter 5

GiBUUToStdHep output file format.

The output format is based around a ROOT tree where each event contains a stdhep particle stack of incoming and nuclear-leaving particles. Some extra generation information is also included, as well as a best-guess at determining the NEUT-equivalent neutrino interaction mode.

The individual output branches are well documented in the doxygen autodocs below: GiRooTracker.

The correspondence between GiBUU's internal 'prodid' and the converted Neut-equivalent code can be seen in the documentation for: GiBUUUtils::GiBUU2NeutReacCode.

The correspondence between GiBUU's internal particle numbering scheme and the pdg codes can be seen in the documentation for: GiBUUUtils::GiBUUToPDG.

Note: Following the GENIE convention, the struck nucleon is given a `StdHepStatus == 11`.

Note: The target nucleus information is saved at index `1` in the StdHep arrays. The `StdHepPdg` is a nuclear PDG code (100ZZZAAA0) and can be decomposed into target A, and Z like:

```
TargetZ = ((StdHepPdg[1] / 10000) % 1000);
TargetA = ((StdHepPdg[1] / 10) % 1000);
```

Chapter 6

Further analysis of stdhep events

Plotting cross sections

As shown in the Quickstart guide, histograms of event properties must be weighted by each event's `EvtWght`. This weighting does most of the leg-work in scaling the event-rate-like distribution to a flux-averaged cross section. Unlike many other generators, the flux averaging has already taken place as part of the simulation and subsequent processing, there is no need to scale by the number of generated events. For a more complete description of how the GiBUU event weights are calculated and used, see `here`.

The final step in producing a flux-averaged differential cross section from the event vector is to divide each bin by it's width. This will result in a binned cross section prediction in units of `10^-38 cm^2 nucleon^-1 <Property Units>^-1`.

An example of this process is shown below:

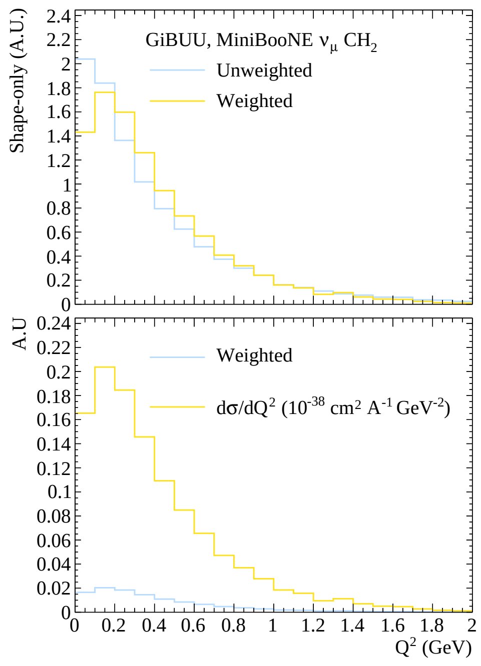

Figure 6.1: Example effect of correctly weighting histogram fills with the EvtWght, and then dividing by binwidth to produce a differential cross section prediction in Q^2

Running event selections

Most analyses will want to post process the particle stack, or make selections, or build composite properties. The reader is directed to the NUISANCE project and the 'GenericFluxTester' tool therein which makes writing such an analysis simple.

As a rudimentary example of such post processing, execute `$ Select_CC1pip_GiBUUStdHep <input_GiBUU.stdhep.root>` passing a produced GiBUU stdhep file. Herein, one produced by executing `$ Generate_MiniBooNE_numuCC_CH2_Events_GIBUU 5` will be used. This script will select CC numu events that contain a single positively charged pion and write out a TTree of the squared four momentum transfer to a file name `CC1pip_Q2_<input_GiBUU.root>`. To check that it looks sensible, we can compare to the MiniBooNE data (which is dumped to the current directory by the script if using a MiniBooNE event vector):

```
root -l CC1pip_Q2_MiniBooNE_CH2_numuCC.root
[root] TH1 *Q2 = new TH1D("Q2",";Q^{2} (MeV^{2});d#sigma/dQ^{2} (cm^{2} MeV^{-2})",20,0,2);
[root] Q2->Sumw2();
[root] CC1PipQ2->Draw("Q2 >> Q2","EvtWght");
// Scale to MeV
[root] Q2->Scale(1E-6);
// Scale to a differential xsec per CH2
[root] Q2->Scale(14.E-38,"width");
[root] Q2->GetYaxis()->SetRangeUser(0,60E-45);
[root] Q2->Draw("E1");
[root] TGraph comp("MiniBooNE_1DQ2_numu_CC1pip.dat");
[root] comp.Draw("L SAME");
```

This is contained within a a `cint` macro and can be executed as `root -l CC1pip_Q2_MiniBooNE_CH2_numuCC.root ${GIBUUTOOLSROOT}/cint_macros/Plot_MiniBooNE_CH2_CC1pip_Q2.C`

The macro used to run the selection and write the analysis tree will be dumped to the current directory when `Select_CC1pip_GiBUUStdHep` is run, but also lives in `${GIBUUTOOLSROOT}/cint_macros/Select_CC1pip.C`.

Note: Before you get worried, most generators underestimate this dataset quite severly. By using the results of the BNL fit parameters for resonant and non-resonant pion production the difference in this dataset can be reduced.

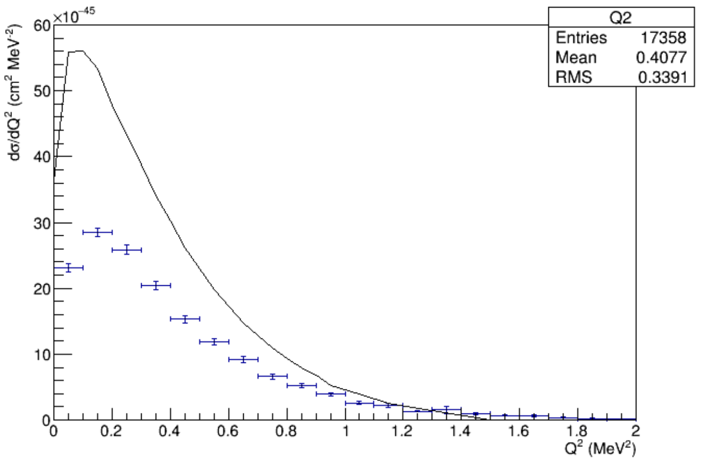

Figure 6.2: Example comparison of generated and selected GiBUU CC1pi+ events to the MiniBooNE data: Solid line is interpolated data, points are the generated prediction

Chapter 7

Namespace Index

7.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

Chapter 8

Class Index

8.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 9

Namespace Documentation

9.1 GiBUUToStdHepOpts Namespace Reference

Options relevant to the GiBUUToStdHep.exe executable.

Variables

- std::vector< std::string > InpFNames

  The location of the input files which was produced by GiBUU.
- std::string OutFName

  The name of the output root file to write.
- bool HaveStruckNucleonInfo
- std::vector< int > nuTypes

  The neutrino species PDG.
- std::vector< int > TargetAs

  The target nuclei nucleon number, A, for the next input file(s).
- std::vector< int > TargetZs

  The target nuclei proton number, Z, for the next input file(s).
- std::vector< bool > CCFiles

  Whether input events for the next file(s) are simulated NC interactions.
- std::vector< float > FileExtraWeights

  An extra weight to apply to the events of the next file(s).
- std::vector< float > NFilesAddedWeights

  An extra weight to applied which averages over the number of files added.
- float OverallWeight = 1

  An extra weight to apply to all parsed events.
- bool HaveProdChargeInfo = false

  Whether the GiBUU output contains the neutrino-induced hadronic particles charge.
- std::vector< std::pair
  < std::string, std::string > > **FluxFilesToAdd**

9.1.1 Detailed Description

Options relevant to the GiBUUToStdHep.exe executable.

9.1.2 Variable Documentation

9.1.2.1 std::vector<bool> GiBUUToStdHepOpts::CCFiles

Whether input events for the next file(s) are simulated NC interactions.

Note

Set by ‘GiBUUToStdHep.exe ... -N ...’

9.1.2.2 std::vector<float> GiBUUToStdHepOpts::FileExtraWeights

An extra weight to apply to the events of the next file(s).

Useful for building composite targets.

9.1.2.3 bool GiBUUToStdHepOpts::HaveProdChargeInfo = false

Whether the GiBUU output contains the neutrino-induced hadronic particles charge.

Note

Assumed true.

9.1.2.4 bool GiBUUToStdHepOpts::HaveStruckNucleonInfo

Whether the GiBUU output contains struck nucleon information.

Note

Assumed true.

9.1.2.5 std::vector<int> GiBUUToStdHepOpts::nuTypes

The neutrino species PDG.

Note

Set by ‘GiBUUToStdHep.exe ... -u xx ...’ Required.

9.1.2.6 float GiBUUToStdHepOpts::OverallWeight = 1

An extra weight to apply to all parsed events.

Useful for building composite targets.

9.1.2.7 std::vector<int> GiBUUToStdHepOpts::TargetAs

The target nuclei nucleon number, A, for the next input file(s).

Note

Set by ‘GiBUUToStdHep.exe ... -a xx ...’ Required.

9.1.2.8 std::vector< int > GiBUUToStdHepOpts::TargetZs

The target nuclei proton number, Z, for the next input file(s).

Note

Set by 'GiBUUToStdHep.exe ... -z xx ...' Required.

9.2 GiBUUUtils Namespace Reference

Utilities which may be helpful for processing GiBUU specific output.

Functions

- int GiBUUToPDG (int GiBUUCode, int GiBUUCharge=0)

  Converts a GiBUU particle code, with associated particle EM charge information to a PDG code.
- int **PDGToGiBUU** (int PDG)
- std::tuple< Int_t, Int_t, Int_t > **DecomposeGiBUUHistory** (Long_t HistCode)
- std::string **WriteGiBUUHistory** (Long_t HistCode)
- void **PrintGiBUUStdHepArray** (Int_t GiBUUCode, Int_t const ∗const StdHepPDGArray, Long_t const ∗const HistoryArray, Int_t StdHepN, std::string const &indent="")
- std::vector< std::tuple< Int_t, Int_t, Int_t > > **GetGenNParticles** (Int_t Gen, Int_t const ∗const StdHepPDGArray, Long_t const ∗const HistoryArray, Int_t StdHepN)
- size_t **GetNParticleGiBUUCode** (Int_t GiBUUCode, Int_t const ∗const StdHepPDGArray, Long_t const ∗const HistoryArray, Int_t StdHepN)
- size_t **GetNFSParticleGiBUUCode** (Int_t GiBUUCode, Int_t const ∗const StdHepPDGArray, Int_t StdHepN)
- std::vector< Int_t > **GetTwoBodyFSParticles** (Int_t const ∗const StdHepPDGArray, Long_t const ∗const HistoryArray, Int_t StdHepN)
- std::vector< Int_t > **GetTwoBodyFSNucleons** (Int_t const ∗const StdHepPDGArray, Long_t const ∗const HistoryArray, Int_t StdHepN)
- std::vector< std::tuple< Int_t, Int_t, Int_t > > **GetFSDecayPions** (Int_t const ∗const StdHepPDGArray, Long_t const ∗const HistoryArray, Int_t StdHepN)
- std::vector< std::tuple< Int_t, Int_t > > **GetDeltaDecayNucleons** (Int_t const ∗const StdHepPDGArray, Long_t const ∗const HistoryArray, Int_t StdHepN)
- std::vector< Int_t > **GetFSPions** (Int_t const ∗const StdHepPDGArray, Int_t StdHepN)
- std::vector< Int_t > **GetInHistoryResonances** (Int_t const ∗const StdHepPDGArray, Long_t const ∗const HistoryArray, Int_t StdHepN)
- Int_t **PionPDGToNeutResMode** (Int_t pionPDG, Int_t NucleonPDG, bool &Warn)
- int **ResonanceHeuristics** (Int_t const ∗const StdHepPDGArray, Long_t const ∗const HistoryArray, Int_t StdHepN)
- int GiBUU2NeutReacCode (Int_t GiBUUCode, Int_t const ∗const StdHepPDGArray, Long_t const ∗const HistoryArray, Int_t StdHepN, bool IsCC=true, Int_t StruckNucleonPosition=-1, Int_t PrimaryProdCharge=-10)

  Converts a GiBUU interaction code to the corresponding NEUT code where possible.

9.2.1 Detailed Description

Utilities which may be helpful for processing GiBUU specific output.

9.2.2 Function Documentation

9.2.2.1 int GiBUUUtils::GiBUU2NeutReacCode (Int_t *GiBUUCode,* Int_t const ∗const *StdHepPDGArray,* Long_t const ∗const *HistoryArray,* Int_t *StdHepN,* bool *IsCC =* `true`*,* Int_t *StruckNucleonPosition =* `-1`*,* Int_t *PrimaryProdCharge =* `-10`)

Converts a GiBUU interaction code to the corresponding NEUT code where possible.

Sometimes the NEUT code is dependent on the particle produced in the intial interaction. CC:

- 1 : QE
- 2 : 2p2h
- 10 : Single pion background (non-resonant)
- 11 : Delta++ (-11 : Delta- for nubar)
- 12 : Delta+ (-12 : Delta0 for nubar)
- 21 : Multi pion production
- 26 : DIS
- 4 : Higher resonance, charge: -1
- 5 : Higher resonance, charge: 0
- 6 : Higher resonance, charge: +1
- 7 : Higher resonance, charge: +2

NC:

- 30 : Single pion background (non-resonant)
- 31 : Delta0
- 32 : Delta+
- 41 : Multi pion production
- 46 : DIS
- 47 : Higher resonance, charge: -1
- 48 : Higher resonance, charge: 0
- 49 : Higher resonance, charge: +1
- 50 : Higher resonance, charge: +2
- 51 : NCEL proton-target
- 52 : NCEL neutron-target

From `https://gibuu.hepforge.org/trac/wiki/LesHouches`

9.2.2.2 int GiBUUUtils::GiBUUToPDG (int *GiBUUCode*, int *GiBUUCharge* = `0`)

Converts a GiBUU particle code, with associated particle EM charge information to a PDG code.

From https://gibuu.hepforge.org/trac/wiki/ParticleIDs

Note

Returns 0 when encountering an unknown particle. Current codes converted:

- GiBUU : PDG
- 1 : p=2212, n=2112
- 2 : Delta++=2224, Delta+=2214, Delta0=2114, Delta-=1114
- 3 : P11(1440) PDGs: 202212, 202112
- 4 : S11(1535) PDGs : 102212, 102112
- 5 : S11(1650) PDGs : 122212, 122112
- 7 : D13(1520) PDGs: 102214, 102114
- 10 : D15(1675) PDGs : 102216, 102116
- 15: P13(1900) P13(1900) PDGs: n/a
- 16: F15(1680) PDGs: 202216, 202116
- 19 : S31(1620) PDGs: 112222, 112212, 112112, 111112
- 20 : S31(1900) PDGs: n/a
- 21 : D33(1700) PDGs: 122224, 122214, 122114, 121114
- 26 : P31(1910) PDGs: 222222, 222212, 222112, 221112
- 27 : P33(1600) PDGs: 202224, 202214, 202114, 201114
- 32 : Lambda PDG: 3122
- 33 : Sigma PDGs: 3222, 3212, 3112
- 53 : Xi PDG: 3322, 3312
- 56 : Lambda_c PDG: 4122
- 57 : Sigma_c PDG: 4222, 4212, 4112
- 101 : pi+=211, pi0=111, pi-=-211
- 103 : rho+=213, rho0=113, rho-=-213
- 104 : sigma=9000221
- 105 : omega=223
- 110 : K+=321, K0=311
- 111 : K-=-321, K0=-311
- 114 : D PDGs: 411, 421
- 115 : D bar PDGs: -411, -421
- 116 : D star PDGs: 413, 423
- 117 : D bar star PDGs: -413, -423
- 901 : 11
- 902 : 13
- 911 : 12
- 912 : 14
- 999 : 22

Chapter 10

Class Documentation

10.1 GiBUUPartBlob Struct Reference

Public Attributes

- Int_t **Run**
- Int_t **EvNum**
- Int_t **ID**
- Int_t **Charge**
- Double_t **PerWeight**
- TVector3 **Position**
- TLorentzVector **FourMom**
- Long_t **History**
- Int_t **Prodid**
- Double_t **Enu**
- Int_t **ProdCharge**

The documentation for this struct was generated from the following file:

- GiBUUToStdHep.cxx

10.2 GiRooTracker Struct Reference

Struct describing TTree structure for the GiBUU rooTracker-like output

```
#include <GiRooTracker.hxx>
```

Public Member Functions

- GiRooTracker ()

  Costructs a GiRooTracker with default values provided by GiRooTracker::Reset.
- ∼GiRooTracker ()

  Free's owned heap space.
- void Reset ()

  Function to reset an instance of this class to its default state.
- void AddBranches (TTree ∗&tree, bool AddHistory=false, bool AddProdCharge=false)

  Will add the relevant output branches to a given TTree.

Public Attributes

- Int_t GiBUU2NeutCode

  The NEUT interaction mode equivalent of the GiBUU interaction type.
- Int_t GiBUUReactionCode

  The GiBUU interaction type.
- Int_t GiBUUPrimaryParticleCharge

  The charge of the first particle produced in the neutrino interaction.
- Int_t EvtNum

  The event number from the input event vector.
- Int_t StdHepN

  The number of StdHep particles in this event.
- Int_t ∗ StdHepPdg

  The PDG codes of particles in this event.
- Int_t ∗ StdHepStatus

  The StdHep Status of particles in this event.
- Double_t StdHepP4 [kGiStdHepNPmax][4]

  Four momentum for particles in this event.
- Long_t ∗ GiBHepHistory

  GiBUU history array, indices correspond to the StdHep arrays.
- Int_t ∗ **GiBHepFather**
- Int_t ∗ **GiBHepMother**
- Int_t ∗ **GiBHepGeneration**
- Double_t GiBUUPerWeight

  GiBUU reported event weight. Directly related to the cross section assuming this run is in isolation.
- Double_t NumRunsWeight

  An extra weight that needs to be applied due using multiple runs to enhance event statistics.
- Double_t FileExtraWeight

  an arbitrary weight that is passed in at parse time. Useful for combining molecular target constituents.
- Double_t EvtWght

  The total XSec weighting that should be applied to this event.

Static Public Attributes

- static constexpr int **kStdHepIdxPx** = 0
- static constexpr int **kStdHepIdxPy** = 1
- static constexpr int **kStdHepIdxPz** = 2
- static constexpr int **kStdHepIdxE** = 3
- static constexpr int **kGiStdHepNPmax** = 100

10.2.1 Detailed Description

Struct describing TTree structure for the GiBUU rooTracker-like output

10.2.2 Constructor & Destructor Documentation

10.2.2.1 GiRooTracker::GiRooTracker ( )

Costructs a GiRooTracker with default values provided by GiRooTracker::Reset.

Allocates GiRooTracker::StdHepPdg and GiRooTracker::StdHepStatus.

10.2.3 Member Function Documentation

10.2.3.1 void GiRooTracker::Reset ( )

Function to reset an instance of this class to its default state.

Used between fillings to result any values to default.

10.2.4 Member Data Documentation

10.2.4.1 Int_t GiRooTracker::EvtNum

The event number from the input event vector.

Note

This will not be unique between GiBUU runs but should be in a single output file.

10.2.4.2 Int_t GiRooTracker::GiBUU2NeutCode

The NEUT interaction mode equivalent of the GiBUU interaction type.

This is determined from the GiBUU interaction type by GiBUUUtils::GiBUU2NeutReacCode.

Warning

This is not a one-to-one mapping.

10.2.4.3 Int_t GiRooTracker::GiBUUPrimaryParticleCharge

The charge of the first particle produced in the neutrino interaction.

Useful for determinining the charge of the resonance in resonant pion production.

10.2.4.4 Int_t$*$ GiRooTracker::StdHepPdg

The PDG codes of particles in this event.

This is determined from the GiBUU particle number by GiBUUUtils::GiBUUToPDG.

Warning

This is not a one-to-one mapping, e.g. resonances are not uniquely determined by the GiBUU scheme.

10.2.4.5 Int_t$*$ GiRooTracker::StdHepStatus

The StdHep Status of particles in this event.

Status Codes in use:

- 0: Initial state real particle.
- 1: Final state real particle.

The documentation for this struct was generated from the following files:

- GiRooTracker.hxx
- GiRooTracker.cxx

Index