A-B-SIMC

John Arrington

July 10, 2001

1 Introduction.

This is designed to be a primer on running SIMC, as well as a summary of the conventions and details.

SIMC is currently set up to simulate the following processes:

1. Elastic/Quasielastic (e,e'p) from Hydrogen, Deuterium, Carbon, Iron, and Gold. All other targets are easily added if you provide a spectral function. NOTE: The spectral function has an 'absorption' factor. This is an overall weight applied to the spectral function, and takes into account a nominal transparency value, and a correction to the independent particle shell model spectral function. Because the IPSM spectral function is normalized to one, even though some fraction of the spectral function (from the correlation tails), you need to reduce the strength to take into account this missing strength. So if you are measuring transparency, you need to remember that the data/SIMC already has these corrections in SIMC. If you are not, you should remember that the value of the transparency used may not be ideal for you kinematics, and keep this in mind when looking at the normalization for heavy nuclei.

2. (e,e'π) from Hydrogen, Deuterium, and Helium-3. A flag allows production of π^+ from protons or π^- from neutrons. Other nuclei can be added relatively easily, with a user provided momentum distribution. However, there are a few hard coded limits that assume a 3-nucleon breakup. These need to be hunted down and generalized first.

3. (e,e'K) from Hydrogen, Deuterium, and Helium-3. A flag allows you to pick K^+ or K^- production, with a Λ, Σ^0, or Σ^- in the final state (as appropriate). You can

also generate bound Hypernuclear final states. Other nuclei can be added relatively easily, in basically the same way as for pion production.

4. The phase space option. This is not currently implemented. Very early version of SIMC had a phase space option. In this version (and most previous versions), it does not work. It can be implemented fairly easily, if someone decides exactly what they want it to do.

2 Getting SIMC.

First you need to copy over the SIMC distribution. The current version is in SIMC_120899.tar.gz, and can be found in johna or johna/SIMC_120899. It is now also available at http://www.cebaf.gov/ johna. We'll find a proper home for the SIMC distribution in the cdaq account at some point (and maybe even on the Hall C web page). Copy the tarred and gzipped version to the directory in which you want SIMC to reside. Gunzip and untar the file. You may need to edit the makefile. Default settings for running at Argonne on the Alphas or running at CEBAF on the HPs or SUNs are in the existing makefile. To run elsewhere, you will probably need to modify the CERNLIB pointers. At this time, SIMC is being maintained with Alpha/Sun/HP compatibility.

3 Running SIMC.

To run SIMC, you need an input file (kept in the SIMC/infiles directory, see appendix A for details), and a momentum distribution (for pion/kaon production) or a spectral function (for (e,e'p)). After making the code (using gmake) type 'simc' and enter the input file name (and spectral function name, if needed) at the prompts. For batch files (or lazy people like myself) you can use the script run_simc and enter the parameters at the command line: "run_simc <inputfile> <SFfile>". If you choose to produce an Ntuple, it will appear in the simc/worksim directory. A summary of the run (<inputfile>.hist) will appear in the simc/outfiles directory. In addition, simc/outfiles contains <inputfile>.gen which has a fixed set of histograms, and <inputfile>.geni which contains information about where events were lost in the simulation.

4 Getting Answers from SIMC.

Once you have an input file you like and have run SIMC, the next step is getting counts out. There are two important pieces to getting results to compare to data. First, you need appropriate cuts. Not only must the cut be identical to the cuts in your data analysis, but they must be CONSISTENT WITH THE VALUES GIVEN IN THE INPUT FILE. This is important because while SIMC will generate events outside of the delta, Em, etc... limits that you give, the normalization will not be correct. Assuming that your xptar and yptar limits cover the full octagon, you should not need additional xptar/yptar cuts in your Ntuple (although you may want them in order to compare to the analysis). You DO need to apply a cut on hsdelta and ssdelta that is at least as tight as the min/max values you use in the input file (SPedge.e(p).delta.min(max)). Similarly, you will need to apply a cut on Em that is equal to or tighter than the cuts.Em.max value. There will be events in the Ntuple beyond your delta and Em limits, due to multiple scattering, energy loss, radiation, and the 'slop' SIMC adds to the generation regions. However, only part of the phase space that contributes to these regions is generated, and so the normalization will be incorrect.

The second thing you need to do is apply the correct normalization. When looking at the Ntuple you need to apply a weight equal to $weight{\cdot}normfac/nevent$, in order to generate a spectrum representing measured counts. *weight* is an entry in the Ntuple (that contains the cross section, as well as any weight coming from event generation tricks). $normfac$ comes from the <inputfile>.hist file, and contains the normalization information from the target thickness, integrated charge, and generation volume. *nevent* is the number of events in the Ntuple. Applying this weight will give the expected counts for whatever charge you gave in the Ntuple. You will need to apply an additional weight to go from the simulated charge to the real efficiency corrected charge, if it is different.

5 Tuning SIMC.

One of the most important things to remember when running SIMC is that you are usually trying to reproduce as closely as possible the data you've taken. This means not only giving SIMC the correct kinematics, but also applying any offsets in SIMC that are seen in the data, and using the same cuts, corrections, and calculations in SIMC as you use in your analysis. Here is a list of things to check when running SIMC.

1. Correcting for energy loss and raster position. SIMC includes a raster contribution to the beam position. You need to give SIMC the raster pattern and size that was used for your experiment. SIMC can also try to correct the events it reconstructs for the raster position. However, you should only apply raster corrections in SIMC if you are using them in your data analysis. Similarly, SIMC applies energy loss event by event, and then applies an average correction for energy loss at the end. You should use this average correction if you are correcting for energy loss in your analysis. If you don't correct for it in the analysis, don't correct for it in SIMC. In both of these examples, there is a switch you can use to disable the correction. If there are other corrections that you apply in your data analysis (or don't apply), you should make sure that they are handled the same way in SIMC.

2. Apply kinematic/spectrometer/beam offsets. There are several different way to apply 'offsets' in SIMC. You can shift the central kinematics in the input file. You can apply beam and target offsets (adjusting the x, y, and z positions of the generated events), or you can apply spectrometer offsets. The difficult part is applying offsets that are consistent with your analysis. You can shift the yptar distribution using beam/target offsets OR by applying spectrometer offsets. You need to determine which is consistent with the data. For example, if you apply a target position offset, the yptar distribution will change, as will the focal plane distributions. If you apply a spectrometer yptar offset, then your yptar distribution will shift, but your focal plane distribution will not. It is a non-trivial procedure to determine your best offsets, and to apply them to the correct variables, and I will not try to go over it in detail here. I will just note that it is a very important consideration. Finally, if you 'remove' an offset in your data analysis (after the reconstruction), then you should make sure to apply the offset in SIMC, and then correct for it after reconstruction in a way that is consistent with your data analysis.

Two offsets that are often present, but may change with experiment are the SOS out-of-plane offset (the 0.15 degree offset when not leveled), which must be applied to the correct (electron/hadron) arm, and the collimator positions (especially the vertical offset in the HMS, which was large in the old HMS setup, and much smaller for HMS-100), which must be changed in the mc_hms.f code.

3. Setting the options in the single arm monte carlos. You can modify the HMS and SOS models to reflect the trigger and/or analysis you use for your experiment. For example, the parameter scintrig (in mc_hms_hut.f and mc_sos_hut.f) tells how many scintillator planes must be hit in order to form a trigger. The default is 3 (for the 3/4 trigger), but if you ran with 4, you will need to change this in the code (there is no option to change this in the input file at the moment). Note that the hodoscope efficiency is NOT represented in the monte carlo, just the geometry of the planes. Similarly, if you require the calorimeter in the trigger (note that the standard PID trigger does NOT REQUIRE the calorimeter, because either the calorimeter OR the cerenkov is enough), then you will need to turn on the fiducial check at calorimeter in mc_hms(sos)_hut.f. The analysis engine can be run with a fiducial cut on the calorimeter, based on the position of the track projected to the lead glass. If you have this cut turned on in the analysis, then the calorimeter energy will only be calculated if the track passes this cut. Therefore, applying ANY calorimeter cut in the analysis also applies a 'hidden' fiducial cut in the calorimeter. You can represent this by applying a cut in the SIMC Ntuple, or by using the track-based fiducial cut in mc_hms(sos)_hut. See the comments in the code for more detail on these cuts.

4. There is a standard set of variables in the SIMC Ntuple. As a rule, the variable names and units are chosen to be identical to the standard engine Ntuples in order to make comparison easier. The standard Ntuple has all of the tracking quantities first: reconstructed focal plane tracks, reconstructed and generated target tracks, followed by some general kinematic variables. After these variables, there will be a

set of calculated variables that depend on the experiment, one for (e,e'p), and one for (e,e'π) (Note that this is not yet implemented). The goal is to calculate these in the same way as in the analysis, which does not necessarily follow the Hall C unit/sign/coordinate system conventions. You should check in results_write.f and in complete_recon_ev (a subroutine in event.f) that the desired variables are being calculated consistently with your analysis.

5. Most of the target/spectrometer geometry is hardwired in the code. If you use non-standard vacuum windows, target geometry, cryotarget wall thicknesses, you will need to change the appropriate parameters. Most of these are contained in target.f, mc_hms(sos).f, and mc_hms(sos)_hut.f. One important one is the x_off, y_off, z_off for the hms collimator. For some of the pre-NucPi experiments, there was a .5mm vertical offset in the collimator (x_off=+0.496). For NucPi (and presumably Fpi, and any of the HMS-100 runs?), the offset was basically removed (x_off=+0.013). In addition, the z_off has varied (z_off=0 for early (1995) runs, +1.5 cm for 1996 and later runs, and +40.17 cm for the HMS-100 tune. These numbers should be checked versus the latest collimator box survey numbers. The same holds true for the drift chamber (or other detector) offsets.

6. Finally, you need to check that the event generation limits you use are adequate. SIMC takes the generation limits you give it, and increases them to take into account multiple scattering, spectrometer resolution, radiation, etc.... However, it is possible that the range SIMC picks will not be adequate. For example, the delta range is increased by some nominal delta resolution of the spectrometer (slop_param_d_HMS(SOS) in simulate.inc). This is not momentum dependent, and may be too small at low momentum or for extreme kinematics (very large values of delta, very long target lengths, etc...). Therefore, it is a good idea to at least once do a high statistics run where you open up the delta, xptar, and yptar generation limits in order to check that the answer you get (after all cuts) does not change. You should do the same for the Em limits (for (e,e'p)) and the radiation limits. You can check the radiation by setting Egamma_gen_max to a large value. If Egamma_gen_max=0, then SIMC calculates the radiation limits to use. You can find out what limits it chose by looking at the .hist file. Setting Egamma_gen_max to a value larger than the overall limits SIMC calculates will let you test that the radiation limits are being calculated and applied correctly. Finally, the variable dE_edge_test can be used to test the energy ranges. The value of every energy limit used in generation, radiation, etc... is increased by the value of dE_edge_test. With luck, you can increase the generation limits, the radiation limits, and dE_edge_test all at once, and the answer will not change, and you will be done.

6 Special Cases (elastic/phase space/etc...).

While SIMC is designed for coincidence reactions, it is possible to use it for single arm measurements of the standard reactions (by converting the 2nd arm into a pseudo-4π detector), or for phase space (acceptance) calculations by setting appropriate flags and/or modifying the cross sections used. In addition, it is fairly easy to modify SIMC so that it can include multiple weights in the output ntuple, so that you can try different cross section models, without rerunning SIMC. Here are details on running some of these special cases.

6.1 Elastic and Quasielastic inclusive [A(e,e')].

First, turn of the electron arm simulation by setting spect_mode=-2. This means that the hadrons will not be run through the spectrometer model, but the hadrons are still generated. Therefore, you need to set the hadron arm kinematics properly and open up the hadron event generation limits (SPedge.p.*.min/max). I usually use +/-90% for delta, +/-900 mr for xptar and yptar. If you get too close to 100% in delta, you can get numerical errors. You also need to be careful that your yptar limits don't allow events at a scattering angle ≤ 0 degrees. If you allow negative angles, the code will not crash, but may not give the right answer, so keep tan(yptar) smaller than spec.p.theta (with a little room to spare). You should also verify in the ntuple that your hadron events are not being cut off by the hadron SPedge cuts.

Note that while events for elastic scattering are generated using only electron quantities, you still need to set the 'hadron arm' kinematics properly and increase the hadron generation limits. This is because SIMC optimizes things in such a way that you only get the right normalization in the region that you tell SIMC to populate. While the events are generated using electron kinematics only (and ALL associated protons are kept), the radiation limits are set in such a way that it does not take into account hadrons that would end up outside of your generation limits. So make sure you are not cutting into the events with the hadron limits.

6.2 Elastic and Quasielastic [A(e,p)].

Basically the same as for (e,e'), except that you set spect_mode=-1 to turn off the electron arm monte carlo, and have to expand the electron arm phase space. In this case, it is more obvious if you have the correct electron kinematics, because you'll only generate events with in the electron solid angle, and they won't make it through the hadron arm.

6.3 Phase Space Simulations.

Sometimes, when all you want is a single arm or coincidence acceptance function, you may want to do a phase space calculation. Depending on what you are trying to do, this may involve populating uniformly in single arm spectrometer coordinates, or single arm 'physics' coordinates, or with a uniform cross section in either the lab or center of mass for the reaction you are trying to simulate. Because there are different 'phase space' options that people might want to run (and because no one has had a strong interest in one), there is no phase space option in SIMC (the flag exists, but it is not implemented). For a single arm acceptance function, one can run the single arm monte carlo with a simple event generator, or use the existing single arm event generator plus spectrometer model (mc_hms(sos)_single.f). The single arm simulation codes contain less physics, and are not generally as up to date. However, they use essentially the same single arm models (mc_hms(sos).f) as SIMC, so you can compare the hms(sos) subroutines to the SIMC versions to get updates. Coincidence and inclusive phase space simulations can be done by using a flat cross section in the center of mass or lab, as desired, but you need to be careful that you turn off any physics that you do not want included (e.g. radiation, coulomb corrections, etc...). Keep in mind that you have to take the process into account when defining an 'acceptance'. For a coincidence measurement, you need to take into account the overlap of the HMS and SOS acceptances, and you have remember to use a cross section weighted acceptance if you are integrating over variables on which the cross section depends.

6.4 Extracting the SIMC Cross Section - The 'Point Monte Carlo Calculation'

Typically, one extracts a cross section by taking the ratio of counts in the data to counts in SIMC (with identical cuts, simulated luminosities, etc...) times the SIMC input cross section at the kinematics where you will quote the cross section. This takes care of bin center corrections, assuming that the ratio of the data cross section to the SIMC cross section is uniform over the acceptance (i.e. that the SIMC cross section has the correct shape). With this assumption, the central cross section for the data is this ratio of data yield to SIMC yield times the SIMC central cross section. There are two important pieces to this procedure. First, the model must be good enough that the shape reproduces the data over the acceptance. If not, you need to reduce the acceptance with cuts, and/or apply an uncertainty based on how well you know the shape (essentially, how well you are doing the bin centering correction). Second, you must be able to figure out the SIMC cross section at the kinematics where you will quote the measurement.

In many cases (elastic scattering, H(e,e'π), etc...) you can just calculate the cross section given the electron and hadron kinematics. If the cross section that SIMC uses is parameterized in terms of the electron/hadron kinematics for the process you are simulating, you can do this directly. However, often SIMC will take an electron-nucleon cross section, and convolve this with a spectral function (e.g. for A(e,e'p)) or momentum distribution (for A(e,e'π) and A(e,e'K)). In this case, you cannot just plug the kinematics into your model, you need to integrate over the nucleon distribution. One way to do this is to run a 'point spectrometer' monte carlo. Essentially, you run SIMC with the electron and hadron variables used in event generation fixed to give you the kinematics you want (e.g. fixed ν,Q^2,θ_{pq}). SIMC will generate events in this region, integrated over the nucleon distributions. The trick is to set the kinematics correctly, make sure the generation region is small enough that you have an (almost) fixed value of ν, Q^2, etc..., and turn of any corrections that you do not want to be part of the cross section you quote. So if you are quoting the 'measured' cross section, you would run with the normal physics options, but without the raster, spectrometer resolution, energy loss, etc... (depending on exactly what you want to quote), but still including the effect of radiative corrections and coulomb corrections (for example). If you are quoting the 'raw' cross section, you should turn off spectrometers, energy loss, multiple scattering, radiation, coulomb corrections, and anything else that you aren't including in your definition of 'raw' cross section (*e.g.* beam energy spread).

For hydrogen cross sections, you can run a 'point' monte carlo (with radiation, spectrometers, etc... disabled) and compare it to the value you get directly from the model cross section you use. This is a good way to test that you are extracting the true input cross section, and not a cross section that has been modified by coulomb corrections, resolution, or anything else that is included in the simulation normally. For hydrogen, every event should have basically the same cross section, since the kinematics of the detected particles fully determine the kinematics at the vertex, and therefore the measured cross section. So you can get the cross section by just averaging the sigcc values in the output ntuple. You can also extract the cross section by just taking SIMC counts divided by simulated luminosity, target thickness, and the solid angle(s) and/or energy ranges over which you generated events. This is more work, but it is a good test because for nuclei, you will not get the same cross section for each event, and you will have to extract the cross section in this way. Doing it this way for hydrogen lets you check that you are using the correct solid angle, luminosity, etc... by comparing directly to the event-by-event cross section.

7 Debugging Tips.

1. If you are getting no successes, you can set ngen to be a negative number and then SIMC generates ngen tries, rather than ngen successes. This allows the run to finish even if there are not events, which means that all of the output files get written. Look at the outfiles/<inputfile>.hist file to check that the kinematics are correct, the correct process is being simulated, etc.... The code that reads in the input files is sometimes a little sensitive about real vs. integer numbers, and a missing decimal point ('1' vs. '1.') has been known to cause trouble.

2. Set spect_mode=1 in the input file (or the 'extra' input file). This generates events, but does not run them through the spectrometer models. Useful for determining if you electron/hadron kinematics that do not match, or if the event generation

is messing up generation of one of the particles. You can also set spect_mode=1, and turn off radiative corrections (using_rad=0), coulomb corrections (using_Coulomb=0), energy loss (using_Eloss=0), etc... if you suspect that one of these may be causing the trouble.

3. 'johna@anl.gov'

8 Appendix A: The Input File.

The input file is read in using CTP, and for convenience, the input parameters are organized into CTP parameter blocks. Each section describes the variables in a given parameter block, and then gives an example (from an Fpi (e,e'π) input file). SIMC.FLAGS is an old writeup describing the flags, and contains some information not included below. Once I've included all of the information in this section, I'll remove the SIMC.FLAGS.

8.1 parameter block "experiment"

This gives the general parameters of the simulations. Most are fairly self explanatory. Choose either doing_phsp, doing_pion, doing_kaon, or none (for (e,e'p)). For the pion and kaon case, you can choose to produce positive or negative hadrons, and enable decay of the hadron (in which case you must provide the decay length). In all cases you must give SIMC the number of events to generate (number of successes if ngen>0, number of attempts if ngen<0) and the total charge to simulate (for normalization purposes). You can also give a filename for 'extra_dbase_file'. This is useful if you have many different kinematics, but you want to keep all of the general options the same. If you give a value for extra_dbase_file, SIMC will read in this second input file after it's done with the main file.

The following example will generate (e,e'π^+) events (with pion decay enabled) until there are 2000 events that make it through both spectrometers. The normalization factor (normfac) will be calculated so that the properly weighted number of events (see 'getting answers from simc') will represent the counts for 1 microCoulomb of beam on target.

```
ngen = 2000                    ;  POS: # of successes; NEG: # of tries
EXPER.charge = 1.0             ;  total charge (mC)
doing_phsp = 0                 ;  (ONE = TRUE)
doing_kaon = 0                 ;  (ONE = TRUE)
doing_pion = 1                 ;  (ONE = TRUE)
doing_piplus = 1               ;  pi+ or pi- production (ONE = TRUE = pi+)
doing_decay = 1                ;  1=decay ON, 0=decay OFF.
ctau = 780.4                   ;  decay length (cm)
extra_dbase_file='extra_SAMPLE' ;  additional input parameters.
```

8.2 parameter block "kinematics_main"

Here you give the beam energy, and spectrometer momentum settings (in MeV/c), the spectrometer angles (in degrees, sign does not matter). In addition, you give the beam energy spread (in %) and a flag telling if electrons are in the hms (hms_e_flag=1) or in the sos (hms_e_flag=0).

```
Ebeam = 3007.                 ;  (MeV)
dEbeam = 0.05                 ;  beam energy variation (%)
hms_e_flag = 0                ;  e- in HMS if set to ONE (SOS if ZERO)
spec.e.P = 594.               ;  e arm central momentum (MeV/c)
spec.e.theta = 56.51          ;  e arm angle setting (degrees)
spec.p.P = 2326.              ;  p arm central momentum (MeV/c)
spec.p.theta = 10.505         ;  p arm angle setting (degrees)
```

8.3 parameter block "target"

This section contains all of the information about the target. You must provide A, Z, and the atomic mass (in amu) for the target nuclei, and the atomic mass (amu) of the recoil nucleus. The recoil nucleus is defined as the A-1 system for (e,e'p), and the A-2 system for A(e,e'π). For H(e,e'p), H(e,e'π/K), and D(e,e'π/K), you do not need to provide a recoil mass (targ.mrec_amu=0). You must also give the density and thickness of the target. For solid targets, give the target angle in degrees where 0 degrees is normal to the beam, and +ve angles are CCW rotations as seen from above (i.e. the downstream face of the target rotating towards the SOS). For cryotargets, the target angle is forced to be zero, but you must choose what target can to use (targ.can=1 is the beer can, targ.can=2 is the tuna/pudding can). Finally, you can give the target abundancy (in %). This is just used as a scale factor for the total luminosity. NOTE: targ.mrec_amu used to be targ.Arec, but this name was changed because it was very confusing (it needs to be the recoil mass, not the recoil A).

```
targ.A = 1.                   ;  target A
targ.Z = 1.                   ;  target Z
targ.mass_amu = 1.007276      ;  target mass in amu
targ.mrec_amu = 0.            ;  recoil mass in amu (eep=A-1 system,pion=A-2 )
targ.rho = 0.0723             ;  target density (g/cm^3)
targ.thick = 327.500          ;  target thick (mg/cm^2) 4.53cm long
targ.angle = 0.               ;  target angle (for solid target) (degrees)
targ.abundancy = 100.         ;  target purity (%)
targ.can = 1                  ;  1=beer can (fpi), 2=pudding can (nucpi)
```

8.4 parameter block "debug"

You can set any of the debug flags (debug(1)-debug(5)) to one in order to have SIMC dump very large amount of information about each event. The comments give an IDEA of what type of information gets dumped.

```
debug(1) = 0                ;  turns on output from brem.f
debug(2) = 0                ;  into/outa subs
debug(3) = 0                ;  spit out values (init. and main loop).
debug(4) = 0                ;  mostly comp_ev, gen_rad diagnostics.
debug(5) = 0                ;  a bit of everything
```

8.5 parameter blocks "e_arm_accept" and "p_arm_accept"

Here, you define the range in delta, xptar, and yptar that you wish to populate for each spectrometer. As a rule, you shouldn't need to change these much for different kinematics, as long as you start out with cuts that cover the full acceptance (except with the longer cryotargets). Your cuts (especially the delta cuts) do not need to cover the entire acceptance of the spectrometer, but you DO need to make sure that you only look at data within these limits when you analyze the Ntuples. Events will be generated beyond the limits you give here, but the normalization will not be correct outside of these limits. SIMC does increase the limits you give it in order to account for resolution, energy loss, and multiple scattering. Therefore, you should not need to artificially increase your generation limits well beyond the range in which you wish to analyze the data, but it's always wise to do a test with larger limits in order to insure that you don't loose counts. Because these are given for the electron vs hadron arm (rather than HMS vs SOS), you need to swap the SPedge.e.* limits with the SPedge.p.* limits if you go from putting electrons in the HMS to electrons in the SOS (and possibly increase or decrease them, depending on the cuts you will use in the analysis).

In the following example, hms_e_flag was set to zero, and so the electron arm is the SOS. Therefore, the delta range is fairly large, and the 90 mr in plane and 50 mr out of plane are somewhat larger than the SOS octagon. Note that the SOS will accept events well beyond +/-22%. If you want to simulate that data, then you need to increase these limits. However, as long as you are applying a delta cut that is below +/-22%, this is OK.

```
SPedge.e.delta.min = -22.0    ;  delta min (%) (SPECTROMETER ACCEPTANCE!)
SPedge.e.delta.max =  22.0    ;  delta max (%)
SPedge.e.yptar.min = -90.0    ;  yptar min (mrad)
SPedge.e.yptar.max =  90.0    ;  yptar max (mrad)
SPedge.e.xptar.min = -50.0    ;  xptar min (mrad)
SPedge.e.xptar.max =  50.0    ;  xptar max (mrad)
```

8.6 parameter block "beamandtargetinfo"

This block includes beam width (x and y, assuming gaussian distribution), the fast raster pattern and size parameters, and x/y/z position offsets for the beam-on-target spot. All distances are in cm.

```
gen.xwid = 0.008868          ;  beam width - one sigma (cm)  (89microns)
gen.ywid = 0.004235          ;  beam width - one sigma (cm)  (42microns)
targ.fr_pattern = 1.         ;  raster pattern: 1=square, 2=circular
targ.fr1 = 0.1               ;  horizontal size OR inner radius(2)
targ.fr2 = 0.1               ;  vertical size OR outer radius(2)
targ.xoffset = 0.0           ;  target x-offset (cm)
targ.yoffset = 0.0           ;  target y-offset (cm)
targ.zoffset = 0.06          ;  target z-offset (cm) z_tar=nominal+zoffset
```

8.7 parameter block "spec_offset"

Spectrometer x,y,z offsets (cm), and xptar/yptar offsets (mr, although it is in fact a slope and therefore unitless). Note that the xptar and yptar offsets represent true offsets in the position of the spectrometer (such as the 0.15 degree (2.62 mr) offset in the SOS out of plane angle when it has not been leveled on the jacks). They do not represent position offsets used with the matrix elements, just an overall shift in the spectrometer angle. The implementation of these offsets have not been checked carfully, and you may need to look in the code to make sure they are doing what you think they should.

```
spec.e.offset.x = 0.         ;  x offset (cm)
spec.e.offset.y = 0.         ;  y offset (cm)
spec.e.offset.z = 0.         ;  z offset (cm)
spec.e.offset.xptar = 2.62   ;  xptar offset (mr)    !x(y)ptar is slope, so
spec.e.offset.yptar = 0.     ;  yptar offset (mr)    !it's really unitless.
spec.p.offset.x = 0.         ;  x offset (cm)
spec.p.offset.y = 0.         ;  y offset (cm)
spec.p.offset.z = 0.         ;  z offset (cm)
spec.p.offset.xptar = 0.     ;  xptar offset (mr)
spec.p.offset.yptar = 0.     ;  yptar offset (mr)
```

8.8 parameter block "simulate"

AAARRRRRRRGGGHHHHH!!!!!! More details to come.

```
using_rad = 1        ;  (ONE = TRUE)
use_expon = 0        ;  (LEAVE AT 0)
```

```
one_tail = -3          ;  0=all, 1=e, 2=e', 3=p, -3=all but p
intcor_mode = 1        ;  (LEAVE AT 1)
Egamma_res_limit = 1. ;  Egamma resolution limit (MeV)
spect_mode = 0         ;  0=e+p arms, -1=p arm, -2=e arm only, 1=none
cuts.Em.min = 0.       ;  (Em.min=Em.max=0.0 gives wide open cuts)
cuts.Em.max = 200.     ;  Must be wider than cuts in analysis(elastic or e,e'p)
using_Eloss = 1        ;  (ONE = TRUE)
correct_Eloss = 1      ;  ONE = correct reconstructed events for eloss.
correct_raster = 1     ;  ONE = Reconstruct events using 'raster' matrix elements.
mc_smear = 1           ;  ONE = target & hut mult scatt AND DC smearing.
deForest_flag = 0      ;  0=sigcc1, 1=sigcc2, -1=sigcc1 ONSHELL
rad_flag = 0           ;  (radiative option #1...see init.f)
extrad_flag = 2        ;  (rad. option #2...see init.f)
lambda(1) = 0.0        ;  if rad_flag.eq.4 then lambda(1) = {TF}
lambda(2) = 0.0        ;  if rad_flag.eq.4 then lambda(2) = {TF}
lambda(3) = 0.0        ;  if rad_flag.eq.4 then lambda(3) = {TF}
Nntu = 1               ;  ONE = generate ntuples
using_Coulomb = 1      ;  (ONE = TRUE)
dE_edge_test = 0.      ;  (move around energy edges)
use_offshell_rad = 1  ;  (ONE = TRUE)
Egamma_gen_max = 0.    ;  Set $>$0 to hardwire the Egamma limits.
```

---

9 Appendix B: The Output Files.

There are five basic output files. The ntuple goes in the worksim subdirectory and will be named *.rzdat. The outfiles subdirectory has three files: *.hist, *.gen, *.geni. The final file is the stuff that gets dumped to the screen when you run SIMC. This contains some important information, in particular errors and warningst hat you should at least look at.

9.1 worksim/<infile>.rzdat

This is the output ntuple. If you know me, you'll know that the less I say about ntuples (and the programs that use them), the better. I'll just point out that for the default ntuples, we try to use the same naming conventions, coordinate systems, units, and formula as we use in the engine. Of course in the end, everyone does their physics calculations slightly differently, so you'll have to check that your data and SIMC reconstruction is done exactly the same way.

9.2 outfiles/<infile>.hist

This is the big input file. It contains the kinematics, and all of the important flags and parameters for the run. If there is something missing that you feel is important, please let me know and I'll add it. The idea is that from the .hist file, you should basically be able to reproduce the input file you used (though it may not be easy). There are a couple of quantities I would like to point out, as they are particularly useful or especially confusing.

$CENTRAL.sigcc$ is the cross section for a central event. This is for an electron along the central ray of the electron arm, and the hardon along the central ray of the hadron arm, both at delta=0kinematics, but it's going to spit out some kind of cross section in any case. Buyer beware. Also note that it may not actually be a true cross section, but the *sigcc* variable returned from the physics.f routine.

$AVERAGE.sigcc$ is in fact not the average sigcc. It is the average weight for the events. For kaon/pion electroproduction, sigcc and weight are very close to each other (just small geometry and radiative correction modifications). For (e,e'p), the weight also includes the spectral function weighting. As a final note, since sum of counts is the sum of $weight \cdot normfac/nevent$, and $AVERAGE.sigcc$ is $weight/nevent$, the total number of events you would get if you look at the ntuple with NO CUTS is $AVERAGE.sigcc \cdot normfac$.

INTEGRATED WEIGHTS (displayed as *wtcontr*) is the total number of counts, and is equal to $AVERAGE.sigcc \cdot normfac$. However, this is not the best number to use, because the predicted counts is only reliable within the spectrometer generation limits you give SIMC in the input file. What I plan on doing is modifying *wtcontr* so that it gives the number of counts for all events within the generation limits. When I apply this cut, I will update the .hist file to say that those cuts have been applied. However, even now it gives a good general idea of then number of counts.

9.3 Output to the screen.

Saving this to a file and checking it after the run is an excellent idea. It will tell you what input file was used and what process it is simulating. More importantly, it will have warning and error messages. Most common are ctp errors (unregistered variable warnings) when you have a variable in the input file that it does not recognise. If you have a typo in your input file, it will create a variable corresponding to what you typed in, and assign it the value you gave it, but this will be totally ignored by the fortran code. A warning of this kind means that one of the input file variables was not set, and while SIMC may still run fine, it may generate garbage results. Also, SIMC will warn you of unusual options, and let you know if it decides to override any of the flags or parameters you give it. It is often hard to check all of this when you run SIMC, because under some conditions you will get hundreds of error messages during the event generation. It only shows these for the first 5000 events, but this can be plenty to push the important messages far off screen.

10 Appendix C: SIMC Coordinate System and Units.

SIMC almost always uses the standard Hall C coordinate system, as defined in the Hall C Vade Mecum[2]. This means that in all cases, $\hat{x}$ points downwards and $\hat{y}$ points to the left when looking along $\hat{z}$. $\hat{z}$ is along the central ray of the spectrometer when talking about hms or sos quantities, and is along the beamline when talking about scattering angles in the lab. θ is the scattering angle (angle between the beamline and the outgoing event), and ϕ is the azimuthal angle. $\tan(\phi) = y/x$, which means that $\phi = 0$ is down, $\phi = \pi/2$ is beam left, etc.... Thus, $\phi^{SOS}_{spec} = \pi/2$, and $\phi^{HMS}_{spec} = 3\pi/2$.

There are, of course, a couple of exceptions. When generating the interaction point, we use the Accelerator coordinate system (as we do in the engine for the BPMs and Raster). In this case, y is vertical (+y is up??), x is horizontal (+x is left??), and z is downstream. In addition, some experiment specific quantities are defined (or calculated) using another coordinate system. Where this is the case, the code is commented, and a transformation is made between the standard SIMC coordinates and the desired values. However, any global variables SIMC uses should use the standard coordinate system.

Units are another matter entirely. As a rule, all distances are in cm, and most energies are in MeV. After that, it's anybody's guess. In order to make the code less unfriendly, I have tried to make sure that all input or output has comments that give the units, and have made all of the Ntuple variables follow the Hall C convention for units (GeV/cm/radians) as well as the standard naming conventions for the engine ntuples.

11 Appendix D: Event Generation.

Let's start with some comments lifted from the subroutine generate:

```
! Generated quantities: (phase_space NOT YET IMPLEMENTED).
!
! phase_space: Generate electron E,yptar,xptar and hadron yptar,xptar??
! doing_hyd_elast: fixed Em, generate electron angles
! doing_deuterium: fixed Em, generate electron fully and proton angles, calc Ep
! doing_eep, A>2: generate electron and hadron energy and angles (calc Em/Pm).
! doing_pion: fixed Em, generate electron energy/angles, p_fermi,
!       hadron angles
! doing_kaon: as doing_pion.
!
! The above is summarized in the following table:
!
!                    ELECTRON                   HADRON
!               ------------------       ------------------
```

```
!               E       yptar   xptar   E       yptar   xptar   p_fermi
!
!H(e,e'p)               X       X
!D(e,e'p)       X       X       X               X       X
!A(e,e'p)       X       X       X       X       X       X
!----------------------------------------------------------------------
!H(e,e'pi/K)    X       X       X               X       X
!A(e,e'pi/K)    X       X       X               X       X       X
!----------------------------------------------------------------------
!phase_space    X       X       X       ?       X       X
!
! So our procedure is the following:
! 1) Always generate electron yptar and xptar
! 2) generate hadron yptar and xptar for all cases except H(e,e'p), D(e,e'p)
! 3) generate p_fermi for D(e,e'p), and D/He pion production
! 4) generate electron E for all but hydrogen elastic and deuterium.
! 5) Generate hadron E for A(e,e'p)
! 6) Set missing energy for cases where it is hardwired
!
! After we generate xptar/yptar/energy, we calculate physics angles (theta/phi),
!  momenta, unit vectors, etc... here and/or in complete_ev.
!
! Note that there are also jacobians associated with some and/or all of
! the above.
! 1: We generate uniformly in xptar/yptar, not theta/phi.  We define the
! phase space volume (genvol contribution) as the product of the xptar/yptar
! range, and have a jacobian for each event taking into account the mapping
! between the solid angle on the unit sphere, and the dxptar/dyptar volume
! (the jacobian is 1/cos**3(dtheta), where dtheta is the angle between the
! event and the central spectrometer vector
! 2: For the D(e,e'p), we take Em as fixed in order to calculate the proton
! energy.  There is a jacobian ( |dEp'/dEm| ).  It comes from integrating
! over the energy conservation delta function: delta(E_D - E_p - E_n - Em).
```

OK, now that that's all clear, the following sections will go over the assumptions and calculations for each process. The following describe the steps necessary to generate the electron and hadron four-vectors at the vertex. Using these values, the physics scattering angles are calculated (see geometry.ps in the simc/documents directory), along with $\nu, \vec{q}, Q^2, E_m, T_{rec}$, and any other desired physics quantities for the specified physics process.

11.1 H(e,e'p)

H(e,e'p): doing_hyd_elast

-Generate electron scattering angles.

-Calculate electron energy from elastic kinematics: $E_{e'} = \frac{E_e \cdot M_p}{M_p + E_e * (1 - \cos\theta_e)}$.

-Calculate q: $\vec{q} = \vec{e} - \vec{e'}$.

-Calculate proton energy/angles: $\vec{p} = \vec{q}, E_p = \sqrt{p^2 + M_p^2}$.

11.2 D(e,e'p)

D(e,e'p): doing_deuterium

-Generate electron scattering angles and energy, and proton angles.

-Fix missing energy and calculate proton energy.

-Apply Jacobian to weight for the event: $\frac{dE_p}{dE_m}$.

-Calculate P_m, and apply weight to the event equal to the value of the spectral function at the generated E_m, P_m.

11.3 A(e,e'p) - A>2

A(e,e'p): doing_heavy

-Generate electron angles and energy, proton angles and energy.

-Calculate Em, Pm, and apply spectral weight.

Summary of generation limits, cuts, etc...

11.4 H(e,e'π)

H(e,e'π): doing_hydpi

-Generate electron energy and angles, pion angles.

-Calculate pion momentum.

11.5 D(e,e'π) and He(e,e'π)

D(e,e'π), He(e,e'π): doing_deutpi, doing_hepi

-Generate electron energy and angles, pion angles, and initial (struck) nucleon momentum.

-Calculate pion momentum.

11.6 complete_recon_ev

The routine complete_recon_ev calculates all of the desired physics quantities for the reconstructed event. This is the place to put experiment specific calculations of physics quantities (or directly in results_write, for simple calculations). For each event, the spectrometer quantities must be passed to the routine (through the 'main' structure), and the following quantities are calculated:

-The unit vectors for the outgoing electron and hadron: $\hat{p'_e}$ and $\hat{p_p}$.

-The virtual photon quantities: $q, \hat{q}, \omega, Q^2$, and ϵ.

-The angles between the virtual photon and the outgoing hadron (following the pion production conventions - should the ϕ_{pq} definition be different for (e,e'p)?): θ_{pq} and ϕ_{pq}.

-Missing momentum and it's components (parallel/perpendicular/outofplane for physics, x/y/z for Heepcheck): $P_m, P_m^{Par}, P_m^{Per}, P_m^{Oop}, P_m^x, P_m^y, P_m^z$.

-Invariant mass: $W = \sqrt{M_A^2 + 2M_p\omega - Q^2}$.

-FOR (e,e'p) ONLY: Missing energy and recoil momentum:E_m, T_{rec}. $T_{rec} = \sqrt{P_m^2 - M_{rec}^2}$, and $E_m = E_e + M_p - E_{e'} - E_p - T_{rec}$.

12 Appendix E: Radiative Corrections.

See the writeup "Radiative Corrections - The SIMC Way"[1].

13 Appendix F: Cross Section Models.

See the seminal work by D. Gaskell[3] for pions. An earlier introduction to some of the problems in pion production for A>1 can be found in ref [4]. The Proton cross section model is DeForest. More details to come (or in Dipankgar's thesis).

References

[1] D. Dutta, "Radiative Corrections - The SIMC Way.", Internal Hall C Note, included in simc/documents.

[2] Hall C Physics Vade Mecum.

[3] D. Gaskell, "Pion Production in SIMC", Internal Hall C Note, included in simc/documents.

[4] D. Gaskell and J. Arrington, "Implementation of the Pion Cross Section Model for E91-003", Internal Hall C Note, included in simc/documents.