Input/Output/Flux/Geometry/FSI in NuWro

Cezary Juszczak

Contents

1. Give an overview of your flux/geometry driver functionality and input formats
2. Are there any structural problems that would be needed to solved to use a common flux/geometry driver
3. Review the output formats of your generator
4. Are ISI and FSI done at separate stages in your generator. If not, why not? How difficult, mechanically, would this factorization be?

NuWro input format overview

```
nuwro [-i <input>] [-o <output>] [[-p "<param line>"]...]
```

- `<input>` is input text file (default is `params.txt`)
- `<output>` is output root file (default is `eventsout.root`)
- every `<param line>` is "appended" to `<input>` file

Input file lines and `<param line>` have the same syntax:

```
# This is a comment line
@this_file_is_included.txt
param1 = value  // sigle value parameter
# Multi value paramter:
param2 = val1 val2 val3 ...
# Multi line parameter:
param3 = line 1
param3 += line 2
...
param3 += line n
```

Beam types in NuWro (`beam_type` choices)

Beams of types 0, 1, 5, and 6 model fluxes homogeneous in space, with all particles moving in the same direction.

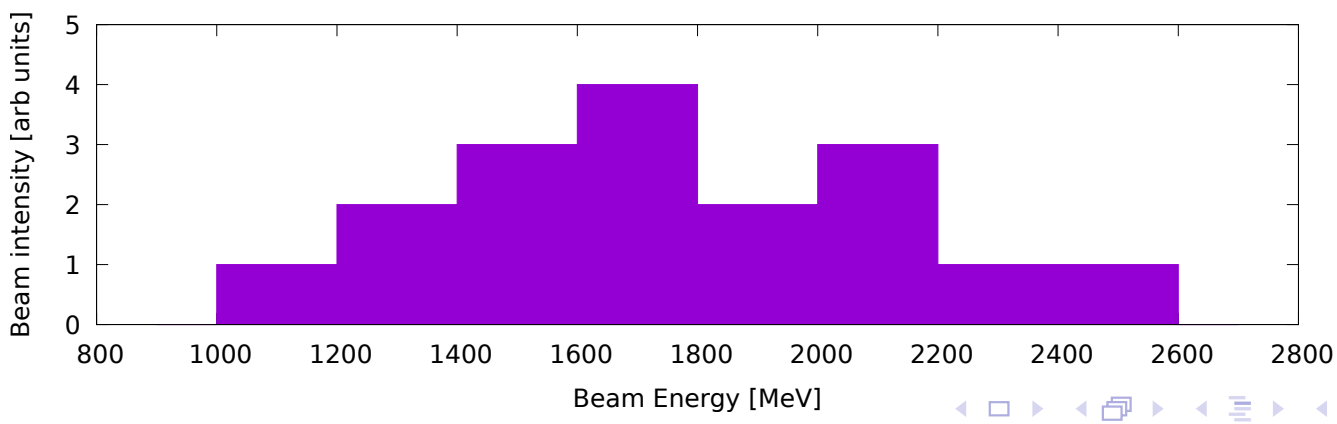
Beam types 2 and 3 are suitable for near detector (eg. ND280). They model fluxes inhomogeneous in space and momentum.

- `beam_type = 0` - single flavor beam
- `beam_type = 1` - weighted mixture of beams of type 0
- `beam_type = 2` - individual neutrinos read from files created by beam simulation software and placed in `beam_folder`. Implemented for ND280.
- `beam_type = 3` - inhomogeneous in space and momentum beam based on multidimensional histograms.
- `beam_type = 4` - build histograms for use with beam type 3 from input files of beam type 2 (works only for ND280).
- `beam_type = 5` - same as 0 but energy profile read from a `THist` object saved in root file
- `beam_type = 6` - same as 1 but energy profiles read from `THist` objects saved in root files

`beam_type = 0` // single flavor beam

Single favor unidirectional and spatially homogeneous beam.

- `beam type = 0` // Beam with a single neutrino flavor
- `beam_particle = 14` // Neutrino pdg code
- `beam_direction = 0 0 1` // Beam direction (z-axis)
- Fixed energy beam:
  `beam_energy = 1000` // $E = 1000$ MeV
- Uniform energy distribution:
  `beam_energy = 1000 2000` // $1000\ \text{MeV} < E < 2000\ \text{MeV}$
- Nonuniform energy profile. Encoded in list of numbers: E_{min} E_{max} and list of bin heights (in arbitrary units) e.g.
  `beam_energy = 1000 2600  1 2 3 4 2 3 1 1`



`beam_type = 1` // Mixture of single flavor beams

Intended for mixed flavor beams. Parameters are:

- `beam type = 1` // Weighted mixture of single flavor beams
- `beam_direction = 0 0 1` // Beam direction for all flavors
- `beam_content` // parameter has many lines.
  Each line of `beam_content` contains three "parameters": `beam_particle`, percentage, and `energy_profile` e.g.:

```
beam_content = -14 100% 0 7600 2.157 ...
beam_content += 14 18.5191% 0 7600 2.298 5.903
beam_content += -12 0.432435% 0 7600 1.233 4.476...
beam_content += 12 0.229053% 0 7600 6.084 18.47...
```

  The percentages do not need to add up to 100%, since only relative values are important.

  The RHS of the % sign has exactly the same meaning as `beam_energy` parameter for beams of type 0.

  The `+=` operator is necessary to append subsequent lines to the parameter value.

`beam_type = 2` // use the flux from a MC output

- Inhomogeneous in space and momentum
- Suitable for use with near detector geometry (e.g. ND280)
- Individual weighted neutrinos read from MC files placed in a folder. Parameters:
- `beam_type = 2` // Must be set to use this mode
- `beam_folder = ~/flux/` // folder with files to be read
- `beam_offset = 0 0 0` // center of coordinates of the beam expressed in the coordinates of the detector
- `beam_file_first = 1` - number of file to start with
- `beam_file_limit` - number of files to be read (0 no limit)
- `beam_weighted` - generate weighted (1) or unweighted (0 - default) neutrinos.
- File format and POT calculation is specific to T2K/ND280

On NuWro startup neutrinos are read from files into memory. Neutrinos not hitting the detector are discarded but accounted for.

`beam_type = 3` // beam based on Multi Dimensional Histogram

- The beam of type 2 has the disadvantage that it can run out of neutrinos, and start to loop thus creating many events with exactly the same neutrinos.
- So it is plausible to make histogram based on the MC simulation files and use the histogram instead of the original list of neutrinos from simulations.
- It is done by running NuWro with the parameter
  `beam_type = 4 // create beam histogram`
  and the parameter `beam_folder =` the path to the beam files that should be included in the histogram.
- After the histogram is created (the `histout.txt` exists) it is enough to set
  `beam_type = 3` // use histogram based beam.
  to use it in simulations.

`beam_type = 5 // Single flavor using root histogram`

This option was created by Patrick Stowell / Luke Pickering.
It has the same functionality as `beam_type=0` except the energy profile histogram is not inlined but read from a root file.
Parameters are:

- `beam_type = 5 // Single flavor from root histogram`
- `beam_particle = 14 // neutrino PDG code`
- `beam_direction = 0 0 1 // along z-axis`
- `beam_inputroot = <file>` // name of the root file
- `beam_inputroot_flux = <name>` // name of the `TH1D` histogram object inside the root file

`beam_type = 6` // Multi favor using root histograms

This has the same functionality as `beam_type=1` except the histograms are read from a root file, and must have the same scale. Probability of choosing specific neutrino flavor is proportional to the total of the corresponding histogram. Parameters are the following:

- `beam_type = 6` // Multi flavor beam from root histograms
- `beam_direction = 0 0 1` // common to all flavors
- `beam_inputroot = <file>` // name of the root file
- `beam_inputroot_nue` // histogram name for ν_e
- `beam_inputroot_nueb` // histogram name for $\bar{\nu}_e$
- `beam_inputroot_numu` // histogram name for ν_μ
- `beam_inputroot_numub` // histogram name for $\bar{\nu}_\mu$
- `beam_inputroot_nutau` // histogram name for ν_τ
- `beam_inputroot_nutaub` // histogram name for ν_τ

Target types in nuwro

- `target_type = 0` // Single isotope
- `target_type = 1` // Mixture of isotopes
- `target_type = 2` // Detector geometry

Detector geometry overview

Detector geometry is read from the root file (e.g. `ND280.root` or `Minerva.root`). It can be used with detector specific beam. The input parameters are:

- `geo_file` - name of the file containing root geometry object
- `geo_name` - name of the geometry object (`TGeometry`)
- `geo_volume` - name of the top volume
- `geo_o = xo yo zo` of the center of region of interest
- `geo_d = dx dy dz` (half dimensions of the region) The box of interest is

$$(x_o - dx, x_o + dx) \times (y_o - dy, y_o + dy) \times (z_o - dz, z_o + dz)$$

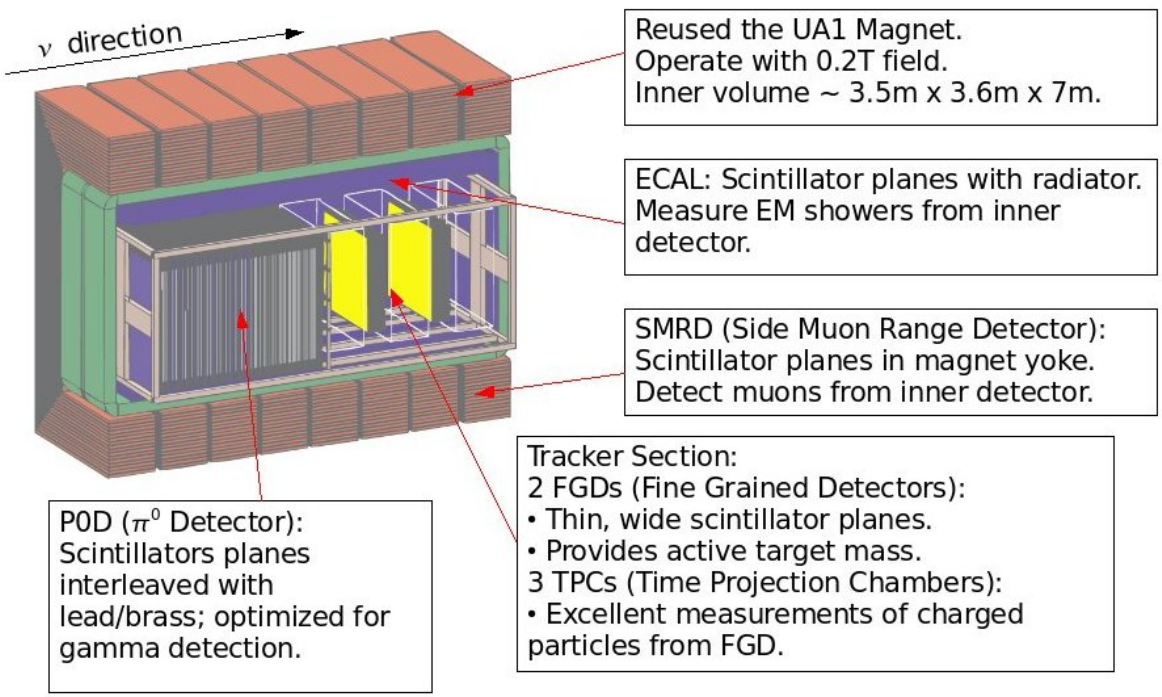
- The only input (not contained in `params.txt`) is the root file given by `geo_name`.

The ND280 geometry can be replaced by any other detector geometry without any changes to the NuWro code.

Detector geometry example ND280

Contents of the file: `data/ND280_975.txt`

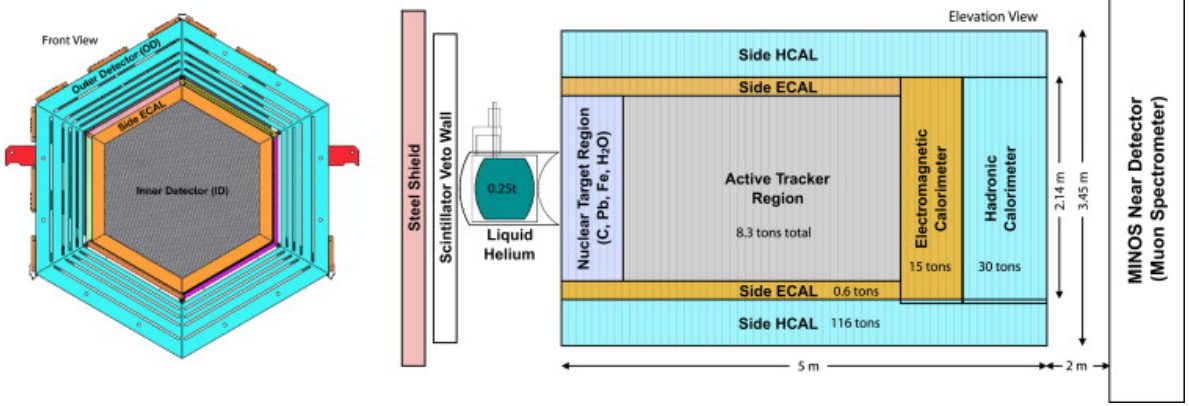
```
# ND280 geometry
target_type = 2  // use detector geometry
geo_file = target/ND280_v9r7p5.root
geo_name = ND280Geometry_v9r7p5
geo_o = 0 0 0 // the center of the box of interest
geo_d = 2000 2000 5000 // its half dimensions
nucleus_target = 2
```



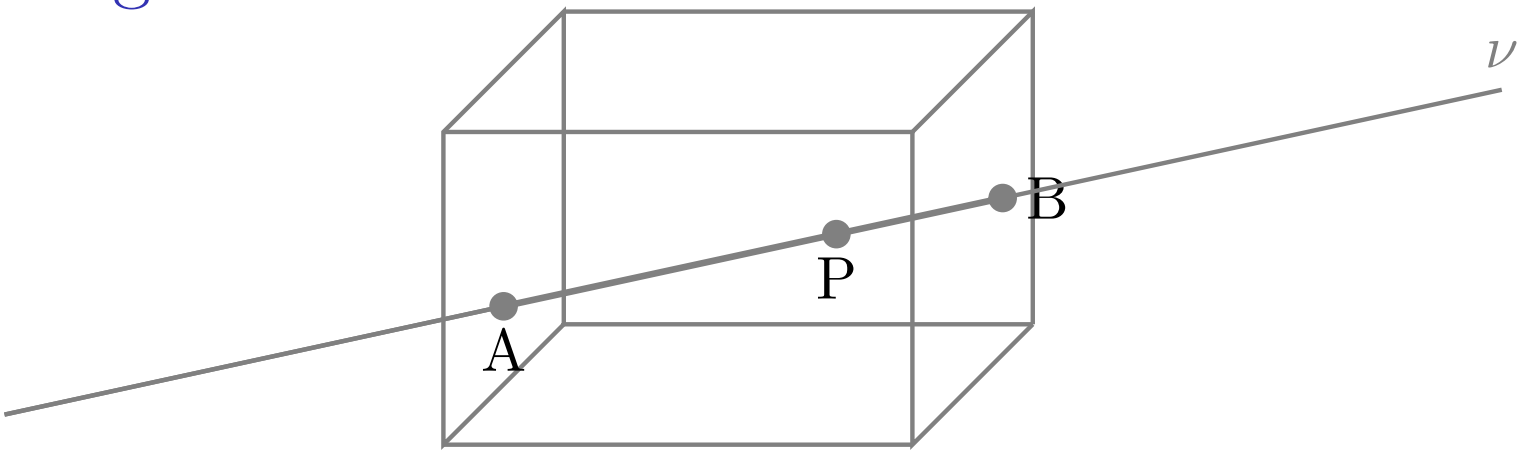
Detector geometry example: Minerva

Contents of the file: `data/Minerva.txt`

```
#Minerva geometry
target_type = 2  // use detector geometry
geo_file = target/Minerva.root
geo_name = Geometry
geo_o = 0 0 750 // center the box of interest
geo_d = 300 300 300 // its half dimensions
nucleus_target=2
```



The algorithm

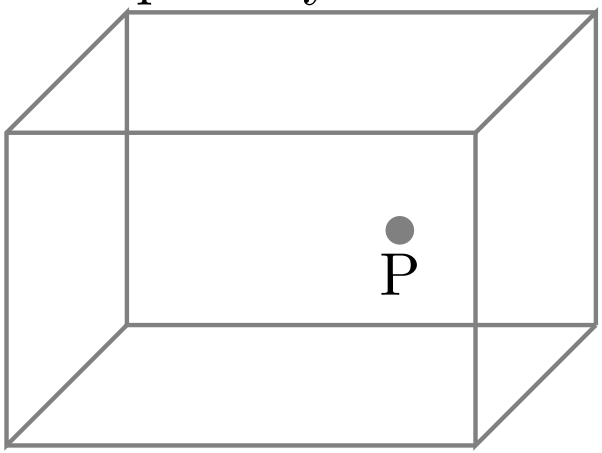


Discard (but calculate %) all ν not crossing the box of interest.
Then repeat:

1. Get ν from the beam.
2. Find entry/exit points A/B and calculate $l =$ length of $\overline{AB}$
3. Take random $P \in \overline{AB}$
4. Calculate $d =$ density of matter at P.
5. Take random $x \in [0, d_{\max} \cdot l_{\max}]$ and go to 1. if $x > d \cdot l$.
6. Get isotope A according its mass share in matter at P.
7. Enter properties of A (p, n, k_F, E_b) in `params` structure.
8. Simulate event for isotope A and decrease its weight by % of discarded neutrinos.

The algorithm (for spatially uniform beams)

For spatially uniform beams the algorithm is even simpler



Repeat:

1. Get ν from the beam.
2. Take random P inside the box of interest.
3. Calculate $d =$ density of matter at P.
4. Take random $x \in [0, d_{\max}]$ and go to 1. if $x > d$.
5. Get isotope A according its mass share in matter at P.
6. Enter properties of A (p, n, k_F, E_b) in `params` structure.
7. Simulate event for isotope A.

Nuwro output files

During its run NuWro creates the following files:

- `eventsout.root` - file with the event objects
- `eventsout.root.par` - file with actual parameters used in the simulation. To repeat simulation type:

  `nuwro -i eventsout.root.par`
- `totals.txt` - text file with effective total cross sections of all active channels
- `eventsout.root.txt` - effective total cross section, its statistic error, efficiency, and number of generator events for each active channel.
- `random_seed` - saved random seed. Mainly for debugging. Enables rerunning `nuwro` with the exactly the same results.
- `q0.txt`, `q2.txt`, `qv.txt`, `T.txt` - differential cross sections calculated during nuwro run. Used by some developers, for quick comparisons when new models are implemented.

The file with events (by default `eventsout.root`) contains all information useful for the analysis.

Structure of the `eventsout.root` file

The `eventsout.root` - content the following objects:

- `xsections` - `TH1D` histogram storing total cross sections of enabled channels.
- `treeout` - a `TTree` with a single branch `e` of `events`. the `event` contains:
  - `params` - input parameters
  - `flags` - event type flags
  - `in`, `temp`, `out`, `post`, `all` - lists of `particles`
  - `weight`, `dyn`, `density`, `pr`, `pn`, ... - scalars, `r` - 3D vector
  - Methods: `q2()`, `nu()`, `N0()`, `q()`, `q0()`, `qv()`, `q2()`, `s()`, `costheta()`, `E()`, `charge(in`, `W()`, `n()`, `f()`, `nof(pdg)`, `fof(pdg)`, `przod()`, `tyl()`, `number_of_nucleon_elastic()`, `number_of_nucleon_ce()`, `number_of_nucleon_spp()`, `number_of_nucleon_dpp()`, `number_of_pion_elastic()`, `number_of_pion_ce()`, `number_of_pion_spp()`, `number_of_pion_dpp()`, `number_of_pion_tpp()`, `number_of_pion_abs()`, `number_of_pion_no_interactions()`, `number_of_nucleon_no_interactions()`, `absorption_position()`, `number_of_jailed()`, `number_of_escape()`, `number_of_interactions()`, `number_of_particles()` ...
- Object oriented approach (including `event` and `particle` methods) simplifies the analysis

Structure of `event` (data)

The `eventsout.root` file contains the `treeout` tree with branch `e` of `events` with the following structure:

- `params` - actual values of all input parameters:
  - `number_of_events` -
  - `number_of_test_events` -
  - ...
- `flags` - booleans flags useful for event filtering
  - `qel`, `res`, `dis`, `coh`, `mec`, `hip` - test if interaction was: (quasi) elastic, resontant, deep inelastic, coherent, meson exchange current, or hiperon production, respectively
  - `nc` , `cc` - true if neutral/charged current event
  - `anty` - true if antineutrino
  - `res_delta` - true if RES pion comes from Delta decay
- `in` - list of `particles` entering primary vertex:
  `in[0]` - neutrino, `in[1]` - nucleon
- `temp` - particles from Pythia6 fragmentation (DIS only)
- `out` - particles leaving the primary vertex, `out[0]` - lepton.
- `post` - all particles (after FSI) which left nucleus
- `all` - all above + intermediate FSI particles

Structure of `event` (data) - continued

- `weight` - cross section (in cm^2) contains:
  - event scross section for files with weighted events,
  - total cross section for unweighted events (default)
- `dyn` - primary vertex dynamics channel number.
- `r` - position of the event inside the detector
- `density` - density of matter at interaction point
- `pr` - number of protons in the residual nucleus
- `nr` - number of neutrons in the residual nucleus
- `r_distance` - distance from nucleus center of absorption point (if happened)
- `res_jacobian`- Jacobian calculated in RES for random kinematics (for reweighting)
- `res_angrew` - store xsec factor coming from angular distribution (for Delta)
- `res_nu` - store neutrino for reweighting
- `res_q` - store q for reweighting
- ...

Structure of `event` (methods)

The `event` methods useful in the output file analysis:

`nu()`, `N0()`, `q()`, `q0()`, `qv()`, `q2()`, `s()`, `costheta()`, `E()`, `charge()`, `W()`, `n()`, `f()`, `nof(pdg)`, `fof(pdg)`, `przod()`, `tyl()`, `number_of_nucleon_elastic()`, `number_of_nucleon_ce()`, `number_of_nucleon_spp()`, `number_of_nucleon_dpp()`, `number_of_pion_elastic()`, `number_of_pion_ce()`, `number_of_pion_spp()`, `number_of_pion_dpp()`, `number_of_pion_tpp()`, `number_of_pion_abs()`, `number_of_pion_no_interactions()`, `number_of_nucleon_no_interactions()`, `absorption_position()`, `number_of_jailed()`, `number_of_escape()`, `number_of_interactions()`, `number_of_particles()`, `nuc_kin_en()`, `num_part_thr(in`, `num_part_thr_withincosine(in`, `num_part_two_thr_withincosine(in`, `proton_cosine()`, `proton_transp_mom()`, `proton_transp_mom2()`, `proton_transp()`, `proton_pair_number1()`, `proton_pair_number2()`, `part_max_mom()`, `part_sec_mom()`, `vert_act()`, `Erec()`, `Q2rec()`, `proton_recoil()`, `neutron_recoil()`, `photon_recoil()`, `meson_recoil_without_masses()`, `meson_recoil_with_masses()`, `lepton_recoil()`, `total_recoil_with_masses()`, `total_recoil_without_masses()`, `neutral_kaon_recoil()`, `proton_max_mom()`, `particle_max_mom(in`, `particle_max_mom_withincosine()`, `particle_max_mom_withincosine_withinmomentum()`, `total_hadr_post()`

They `event` and `particle` methods are available from both native `C++` scripts and `root` scripts.

Structure of `particle` (data)

In NuWro there is no distinction between the runtime `particle` and serialized `particle`.

Each `particle` has the following data:

- `x,y,z,t` - its momentum and energy (p_x, p_y, p_z, E)
- `_mass` - its mass
- `r` - last position inside the nucleus (r_x, r_y, r_z)
- `pdg` - its PDG code
- `ks, origin` - used only by Pythia6 routine
- `travelled` - distance from creation
- `id` - index in the vector `all`
- `mother` - index of mother in the vector `all`
- `endproc` - `id` of the interaction at the track end
- `his_fermi` - its Fermi energy (in the LFG model)
- `primary` - true if it comes directly from primary vertex

Structure of `particle` (methods)

Each `particle` has the following 'reader' methods:

- `E()`, `energy()` - energy
- `Ek()` - kinetic energy
- `m()`, `mass()` - mass
- `mass2 ()` - mass squared
- `charge()` - charge
- `p()` - momentum as a 3-vector
- `p4()` - four-momentum
- `momentum()` - value of the momentum (number)
- `momentum2()` - momentum squared
- `v()` - velocity as a 3-vector
- `v2()` - velocity squared
- `lepton()`, `pion()`, `nucleon()`, `proton()`, `neutron()` - test if particle is: lepton, pion, nucleon, proton, neutron.
- ...

Using particle and event methods

Many interesting histograms may be obtained in just one line:

- `in[0].E()` - Energy of incoming neutrino
- `in[0].r.z` - third coordinate of the primary vertex
- `out[1].momentum()` - momentum of outgoing lepton
- `post.z` - z-coordinate of momentum of outgoing particles
- `r.x, r.y` - x and y coordinates of the event inside the detector
- `@temp.size()` - number of Pythia6 particles
- `@post.size()` - number of particles leaving the nucleus
- `post.Ek()` - kinetic energies of particles leaving the nucleus
- `number_of_interactions()` - number of FSI interactions
- `number_of_pions()` - number outgoing pions.

Output file converters

Currently there exist three file format converters:

- `nuwro2neut`
- `nuwro2nuance`
- `nuwro2rootracker`

More can be readily created, but maintenance of the code could be a problem.

ISI/FSI separation

- Yes, indeed FSI is done separately after ISI is finished
- However, for some channels FSI is omitted (COH, MEC)
- For QEL events with Spectral Function formalism the effective FSI routine is used instead of the default one

Summary

- Input:
  - Plain text input - modularity provided by `@` include sign.
  - The same grammar in files and command line arguments.
  - Where needed the OO input from `root` files is used.
- Output:
  - The main output file contains `TTree` of `event` objects.
  - Inclusion of `event` and `partile` methods in output file simplifies the analysis.
  - Event methods could be standardized across generators.
- Flux:
  - The far detector flux format is easy to standardize
  - The near detector flux driver depends on the structure of the detector specific flux files
- Geometry:
  - The geometry driver and algorithm are easily portable
- FSI:
  - The FSI part is mostly independent of the rest of NuWro.
  - It can be used as a standalone application.