nEXO OD FLUKA Simulations Manual

Regan Ross

April 20, 2023

Abstract

This document is intented to provide insight into the functioning of the nEXO FLUKA simulations and justifications for choices made in their design. Further, this work should provide a near comprehensive baseline for anyone willing to replicate similar cosmogenic studies for the nEXO collaboration, or even another. FLUKA can be a daunting software to work with given it is written in FORTRAN77, has sparse documentation, and whose source code, without a specific licence, is veiled to the user. However, this proprietary simulation program is fast and rich with built-in features that have been used extensively for decades. Moreover, it is particularly useful for the case of studying cosmogenic muons at several hundred GeV energies and their secondaries. This document will provide an overview and explanation of the FLUKA features enabled for the specific nEXO Outer Detector (OD) case and an overview of the additional user routines, scripts, and containerization procedures that are all ancilliary to the base simulations (but required to run the simulations on the SDF cluster and perform the more in-depth analyses required by our questions).

List of Figures

Contents

The FLUKA Input File ('.inp')

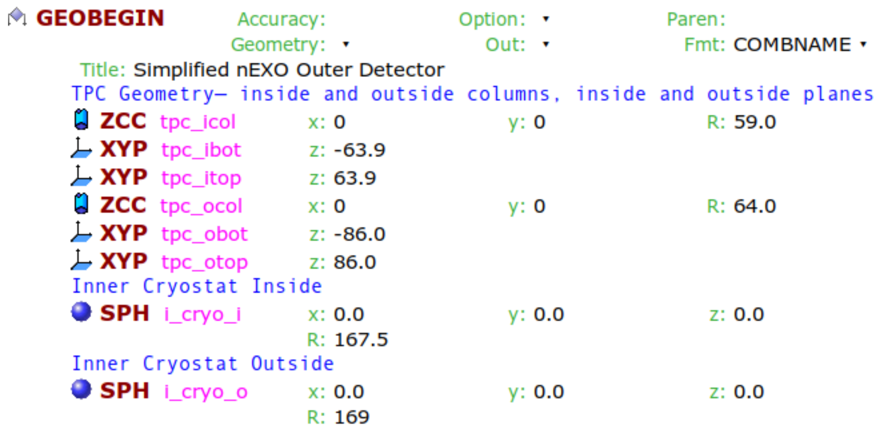
1 Introduction

The input file is exactly as it is named— it is the De facto interface between the user and the FLUKA binaries. It contains, among other things, the entire physical configuration of the media through which the particles are transported, the defining characteristics of the impinging beam, options for enabling particular physical processes, definitions of materials, and cards to deploy default FLUKA scoring methods. It is a human-readable ascii file with a very specific format. Each input card in the input file must not contain more than 8 fields each of which has a character limit. This is due (presumably) to constraints in the early development of FLUKA. Programmed in FORTRAN77 which imposes constrains on the length of statements which, in the early days, were written into computers (with very little memory) using paper punch cards. Now, this formatting is an annoying anachronism but it probably does still keep the program slim and fast. Generally though, a user need not worry about writing the input file directly, as there is a great GUI interface to FLUKA called *flair* which, by the way, is open source and contains all the possible input options. The next section will overview the specific components of the nEXO OD input file and the functions they serve.

2 The nEXO OD Input File

2.1 Geometry

Arguably the most important parts of the input file are the cards defining the configuration space— namely the detector and media with respect to the cartesian coordinate system. In FLUKA this is known as the *geometry*. The geometry section of the input card is demarcated by **GEOBEGIN** and **GEOEND** cards. Between these two cards are first, cards for various *bodies* which are basic 2D and 3D geometric surfaces and second, cards for *regions* whose bounds are defined by combinations of geometric *bodies*. For instance, a triplet of bodies could be a cylinder whose axis lies on the z-axis, and two planes lying parallel the $x - y$ plane at different zs. A region defined by these bodies could be the 3D cylindrical volume made by the cylinder body capped off on either end by the two planes. This is exactly the type of region representing the nEXO OD and the nEXO TPC.



```
GEOBEGIN    Accuracy:            Option: ▾          Paren:
            Geometry: ▾          Out: ▾             Fmt: COMBNAME ▾
   Title: Simplified nEXO Outer Detector
   TPC Geometry— inside and outside columns, inside and outside planes
   ZCC  tpc_icol      x: 0              y: 0              R: 59.0
   XYP  tpc_ibot      z: -63.9
   XYP  tpc_itop      z: 63.9
   ZCC  tpc_ocol      x: 0              y: 0              R: 64.0
   XYP  tpc_obot      z: -86.0
   XYP  tpc_otop      z: 86.0
   Inner Cryostat Inside
   SPH  i_cryo_i      x: 0.0            y: 0.0            z: 0.0
                      R: 167.5
   Inner Cryostat Outside
   SPH  i_cryo_o      x: 0.0            y: 0.0            z: 0.0
                      R: 169
```

Figure 1: An example of the geometry body declarations in the input file

Figure 1 shows the first part of the geometry declarations in the nEXO input file. We see the first card is the **GEOBEGIN** with the argument *fmt* set to COMBNAME. This tells the FLUKA binaries how to read in the following geometry cards. This is not particularly important. This seems to be the default mode. Text in blue indicates a comment (equivalent to a FORTRAN77 comment in the input file) and body variable names are in pink and are limited to lengths of 8 characters. The names or types of cards are fully capitalized and in maroon. The argument names for each card are written in green and the arguments follow with an 8 character length limit. These colour conventions hold for every other type of input card. Each card can have many arguments, but generally, not all are necessary. In the geometry body cards however, each argument is provided as these arguments are all necessary to uniquely define their respective geometric body. For instance, there is a body called i_cryo_i defined with a SPH card. The name is intented to be shorthand for "inner cryostat inside" and, being a sphere, requires a radius and three coordinates for its location in space. There are many choices for bodies in FLUKA, each of which fairly simple to define.

Regions, as previously discussed, are created with logical combinations of bodies. Imagine the bodies defining surfaces in 3D, and the regions being the volumes they surround. Following the set of cards defining the bodies, there is an **END** card, then the region cards as shown in figure 2, finally there is the **GEOEND** card. Each region must be assigned **one** material— we'll get to material assignments later. For example, the region inside the nEXO OD that is full of water would be defined by the OD inside body minus the outer cryostat outside body leaving the configuration of a cylinder with a spherical hole in its volume— the entirety of this region is to be water.

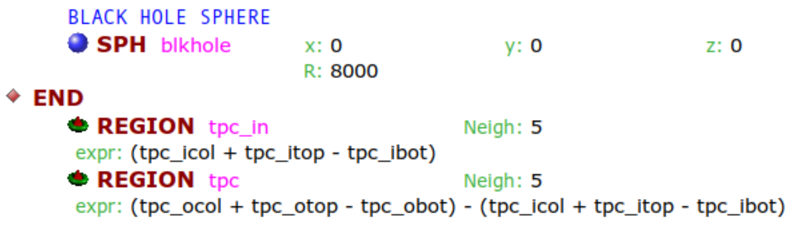

```
BLACK HOLE SPHERE
● SPH  blkhole      x: 0              y: 0              z: 0
                    R: 8000
◆ END
  REGION  tpc_in                Neigh: 5
  expr: (tpc_icol + tpc_itop - tpc_ibot)
  REGION  tpc                   Neigh: 5
  expr: (tpc_ocol + tpc_otop - tpc_obot) - (tpc_icol + tpc_itop - tpc_ibot)
```

Figure 2: An example of the geometry region declarations in the input file

Looking at figure 2, we see the final body declaration followed by the first few region declarations in the nEXO input file. The final body here is important— each FLUKA configuration defines its physical boundaries by assigning material *blackhole* to its outermost surface- hence the name of the sphere. The first region we have defined is tpc_in will be the inside of the TPC cylinder— the part filled with liquid xenon. While no number is assigned by the user to the region, FLUKA assigns it number 1 as it is the first defined- this will be important later. The construction of the region is in the expr argument given by "(tpc_icol + tpc_itop − tpc_ibot)". We first have *tpc_icol* which is the inside cylinder (along z) of the TPC which alone is unbound and spans the entire z-axis. To the cylinder we add *tpc_itop* (xy-plane) which sets the upper bound, and subtract *tpc_ibot* (xy-plane) which sets the lower bound. This process is similar for all other region declarations. Unfortunately, one may not reference regions by name in region expressions, only bodies; hence the following region declaration of the tpc which is intended to be the material composing the TPC. The Neigh argument is an integer which defines "neighbourhoods" relating regions to each other. This can perhaps be deployed for larger, more complex geometries but given the simplicity of the nEXO OD geometry and the lack of thorough documentation of this feature, it has been set to the default value of 5 for each region. Once more, the **GEOEND** card (not shown) brings us to the end of the geometry card section.

2.1.1 The Chosen Configuration

The simulations performed here deployed a simplified geometry of nEXO. There were no PMTs included, no complicated internal structure and no contoured cavernous room around the detector. It is simply a large stainless steel cylinder surrounded by norite rock and full of water containing the inner and outer cryostats and the copper TPC filled with liquid Xenon-136. That's it. The muons are propogated through at least 15m of rock before reaching the detector, and the rock surrounding the OD laterally is 20m, and below it, 5m. Where possible, the measurements for the configuration were taken from the “nEXO Preliminary Design Report” except for the size of the OD which has been adjusted to meet the more recent specification.

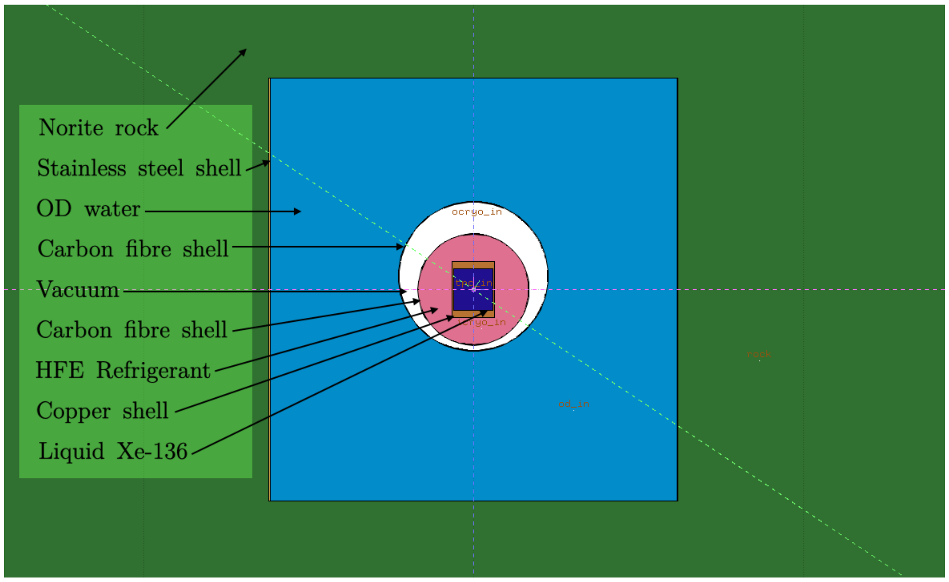
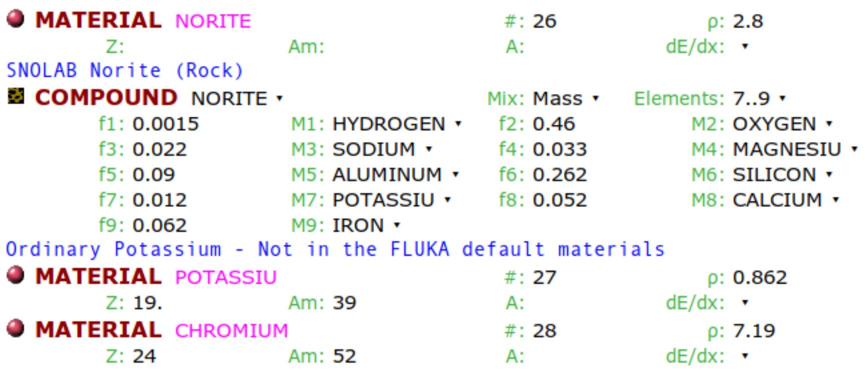


Figure 3: A 2D projection of the nEXO geometry used in the FLUKA simulations

2.2 Media

After *geometry* the most natural section to discuss in the input card is the *Media* section. In this section, materials are defined if they aren’t in the FLUKA default material database and they’re assigned to regions. There are four main types of cards in this section: MATERIAL, COMPOUND, ASSIGNMA, and LOW-MAT.



```
● MATERIAL NORITE                              #: 26            ρ: 2.8
      Z:              Am:                      A:           dE/dx: ▾
SNOLAB Norite (Rock)
▨ COMPOUND NORITE ▾                          Mix: Mass ▾   Elements: 7..9 ▾
      f1: 0.0015      M1: HYDROGEN ▾      f2: 0.46          M2: OXYGEN ▾
      f3: 0.022       M3: SODIUM ▾        f4: 0.033         M4: MAGNESIU ▾
      f5: 0.09        M5: ALUMINUM ▾      f6: 0.262         M6: SILICON ▾
      f7: 0.012       M7: POTASSIU ▾      f8: 0.052         M8: CALCIUM ▾
      f9: 0.062       M9: IRON ▾
Ordinary Potassium - Not in the FLUKA default materials
● MATERIAL POTASSIU                            #: 27            ρ: 0.862
      Z: 19.          Am: 39                   A:           dE/dx: ▾
● MATERIAL CHROMIUM                            #: 28            ρ: 7.19
      Z: 24           Am: 52                   A:           dE/dx: ▾
```

Figure 4: A subset of the media section in the input file

First, referring to figure 4 and describe the first card in the media section. It is a MATERIAL card with variable name NORITE. This card declares a new material called NORITE which is given a density, and a user-chosen material number for later reference [1]. Norite rock is an abundant rock around the Sudbury basin.

[1]The numbers for user-defined materials can’t start at 1, they proceed from the last number of the native FLUKA materials list (25)

It is a mixture of various elements and must therefore be defined with the subsequent COMPOUND card. The arguments for this card are the component materials of the compound and their respective fractions by mass, volume, or atom abundance. These arguments should be very clear, the odd one out is Elements which simply allows for the resizing of the compound card to accomodate more or fewer elements. The following cards, POTASSIU and CHROMIUM are necessary to define as they are not in the default FLUKA materials list. Given that these are elemental, we must provide the arguments of atomic number (Z), the atomic mass (Am) in g/mol, and the atomic mass number (A) which is assumed to be the most abundant for the given (Z) if left unspecified. Of course ρ is the density, and dE/dx allows to select another material to use for the case of ionization— we do not use this feature.

Once all the materials are defined they are then assigned to regions with the ASSIGNMA cards. This card a list of arguments including the material to be assigned and the regions it will be assigned to. One of these cards can assign a material to several regions, however, the regions had to have been declared in some regular order for this to work for multiple regions. That is, a user might wish to have every region from the first to last of N regions to be full of water. In this case, the card would have arguments Reg: $= 1$ and to Reg: $= N$. Alternatively, a user can assign a material to every k^{th} region in the range $[1, N]$ by setting Step: $= 3$. This seems like a strange and uncomfortable way to do things because any modification to the region declarations can totally offset the material assignments. Nevertheless, it is how it works. In the "nEXO_OD.inp" input file, each region is assigned a material separately and by name. This issue ought not occur.

2.2.1 Specific Materials

Now that the input options have been discussed, we will overview some of the materials used in the "nEXO_OD.inp" input file as these may not be an exact representation of the material structure of nEXO. First,

The Muon Source User Routine

The Scoring User Routine ('mgdraw.f')

Interpreting Output

Appendices

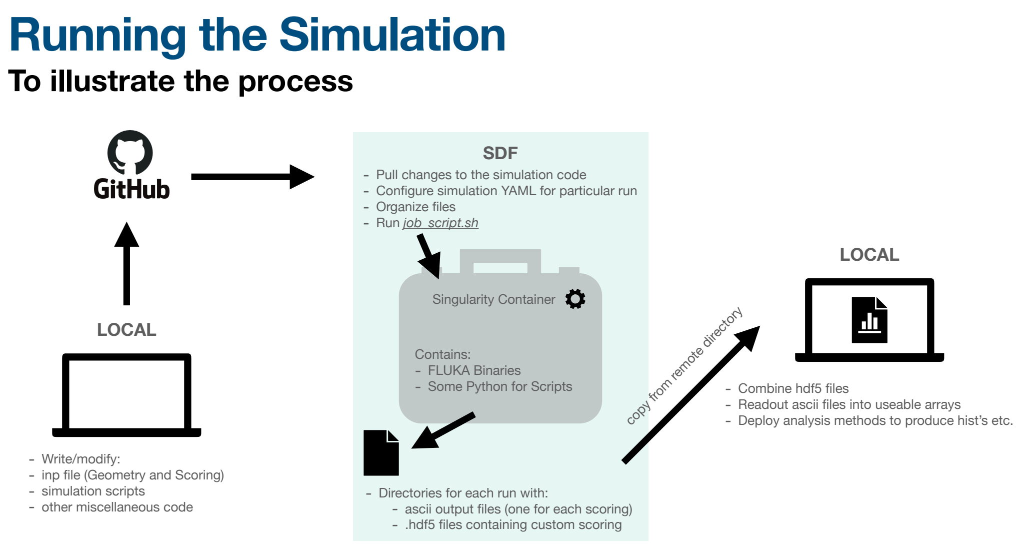

Running the Simulation
To illustrate the process
GitHub
SDF
- Pull changes to the simulation code
- Configure simulation YAML for particular run
- Organize files
- Run job_script.sh
Singularity Container
Contains:
- FLUKA Binaries
- Some Python for Scripts
- Directories for each run with:
- ascii output files (one for each scoring)
- .hdf5 files containing custom scoring
LOCAL
- Write/modify:
- inp file (Geometry and Scoring)
- simulation scripts
- other miscellaneous code
copy from remote directory
LOCAL
- Combine hdf5 files
- Readout ascii files into useable arrays
- Deploy analysis methods to produce hist's etc.

Figure 5: A visualization of the remote simulation and development process

Containerization with Singularity

In order to deploy FLUKA on a remote cluster, SDF for instance, it must be packaged up in a *container*. SDF recommends using Singularity which is essentially designed for containerizing software with its dependencies to maximize portability. Naturally, there are a few caveats to containerizing the simulation. The first, rather inconvenient constraint is that Singularity containers must (at the time of writing) be built from within a Linux OS. Therefore, to build a Singularity image using a Mac, one must use a linux VM. This requires, among other pieces, Vagrant which is used to run the Linux virtual machine. There's no use in giving a comprehensive guide to installing Singularity, Vagrant and creating the entire Singularity image here; these softwares are actively being developed and improved. Readers are referred to the Sylabs Singularity documentation: `https://docs.sylabs.io/guides/3.11/admin-guide/installation.html#mac`.

The definition file "`.def`" for the Singularity container is in the Github repository. Should the reader wish to reproduce or modify the container, they'll need at minimum a CERN FLUKA licence in order to download the FLUKA binaries and they'll need to be placed in the directory where the container will be created along with any other files and software that cannot be installed separately from within the "`.def`" file.