8/29/14

Quick Start Guide for runSimulator.py

A few quick notes on how to use the nEXO background simulation script *runSimulator.py*, as well as the clustering algorithm *clustering.py*. More detailed, proper notes should be made once the simulator is properly developed and all of the sources and locations are properly coded.

Required Files

Root and Geant4 are needed to use *runSimulator.py*. At the moment, the root installation being used is found at:
/data/data030/mbatygov/ROOTs/5.34.17/bin/root
and the Geant4 build being used is found at:
/data/data033/exo/software/nEXO_MC/cluster-trunk-build

The drive on which you are working must also have sufficient space available to it to hold the many root files that may be created during the course of the simulation. The size and quantity of files will vary with simulations.

A clustering algorithm may also be supplied to group nearly collocated energy depositions. Currently *clustering.py* is being used. Even if the clustering flag is set to false, the string variable specifying the path to the clustering algorithm must be set to avoid an error (even if just to an empty string; this nuisance requirement should be fixed in future).

User Specified Variables

These variables are those that should be changed to specify the simulation. They are set in the *Parameter Region* block of the code, at the beginning of the script. No other areas of the file should need to be modified by a user.

PWD:	Present working directory, shouldn't be changed by the user. A convenient variable to have when defining other variables.
PathTonEXOBuild:	Full path to the nEXO build that is to be used to run the simulation.
PathToClusteringAlgoritm:	Full path to the python script to cluster energy depositions that are nearly collocated (or whatever clustering algorithm that might be developed). Even if the ClusterDepsFlag is turned off (see below) this variable must be declared to avoid an error (any string is fine).
HoldingDir:	A directory to hold the many macros, shell scripts, and root files that are created and (possibly) submitted to the computing cluster during the course of the simulation. Many files may be created, so it is recommended that you use an empty directory for this purpose. If the given directory doesn't exist, it will be created.

	Note that the drive on which you are working must have sufficient space to accommodate the root files being created.
PreClustRootFile:	A string identifying the root files produced **before** the clustering algorithm is applied. The root files will have isotope name, location name, and job index appended to the file name.
PostClustRootFile:	A string identifying the root files produced **after** the clustering algorithm is applied. The current clustering algorithm groups deposition from different isotopes, so these root files won't have isotope information in their names, but they will still have region name and job index appended.
IsotopeList:	This is a list of the names of the isotopes you want simulated. For example, if you want to simulate the entire decay chain of some isotope, you must specify all of the daughters in the decay that you want to simulate.

Note: *Some isotope have multiple decay modes with different probabilities for each, and thus the user may want to simulate a different number of events for different isotopes in the chain. The script does not currently have this ability.*

RegionName:	The name of the region in which the event takes place. Possibilities are listed in the *RegionBuilder* function within *runSimulator.py* and is currently incomplete.
NumJobs:	Specifies the number of separate jobs that are to be submitted to the computing cluster. Each jobs occupies one core of the cluster. It is recommended that you split the overall simulations into a number of jobs equal to the number of cores you are permitted to use at a time (assuming this is the only project you are using the cluster for at the time).
NumDecaysPerJob:	Specifies the number of events per job, per isotope. Total number of events in the overall simulation is NumJobs * NumDecaysPerJob * length(IsotopeList).
ClusterDiam:	Specifies the maximum distance at which two energy depositions are considered collocated. Only relevant for this particular clustering algorithm.
ClusterDepsFlag:	Boolean to direct the script to perform the clustering algorithm specified in *PathToClusteringAlgorithm*
UseClusterFlag:	Boolean to direct the script whether to use the computing cluster. If set to 1, the shell script generated is sent to the cluster, otherwise the script is modified in a few ways and can be sourced from the shell. The shell script as well as the macro it runs can be found in *HoldingDir*.

The Region Builder Function
This function is called in the main body of the script to set up the code that the macro needs to specify in which region of the detector that the decay events will take place. The function is a simple if block mimicking a switch block which selected a particular set of lines of code given the name of the region. This is required because not all regions that we may want to specify for decays are hard coded as volumes in the nEXO geometry (although many of them are) and so in those cases the */gps/pos/confine* command cannot be used. This function is not complete, and will need to be modified as certain volumes come in and out of existence as the geometry develops (the inactive region, for example).

The Dictionary Block
This region defines two dictionaries that give the atomic number and weight of all of the isotopes used in simulations. Isotopes can be added or removed in any order at any time.

Environment Variables Block
This block of code adds to the submission script the commands to set up the ROOT and Geant4 environment variables on the core of the computing cluster being used. If *UseClusterFlag* is set to 0, these lines of code are not added. This block may need to change if different ROOT or Geant4 builds are used.

The String Replacement Dictionary
When defining a string in python (and many other languages), the common way that variables are introduced into the string is though format specifiers (%s for string, %d for integer, etc.) For very long strings, remembering the order in which format specifiers were introduced and matching them with the appropriate variables at the end of the string can get impractical. The solution is to be able to pass the names of the variables directly to the string. The way this is done is by introducing a dictionary so that the keys of the dictionary are strings equal to variable names, and the values are the variables themselves. This is the string replacement dictionary. Any variable that you want to be able to put into a string must be added to this dictionary.

The Macro Building and Submission Script Building Blocks
The master script *runSimulator.py* works by creating a shell script and a macro for each job to be run, as well as the corresponding root files once the simulation is finished. E.g. if the simulation is split into 100 jobs with 2 isotopes to be simulated, 100 shell scripts and 100 macro files will be generated, which will in turn generate 200 pre-clustering algorithm root files (100 for each isotope), and if *ClusterDepsFlag* is set to 1, 100 post-clustering algorithm root files. The shell scripts are submitted to various cores on the cluster (or can be executed in the shell) and contain instructions to set up environment variables on the core, as well as to execute the respectively created macro file. If *ClusterDepsFlag* is set to 1, a line to execute the clustering algorithm is also written in the submission script. All created files are stored in *HoldingDir*.