1. Introduction

The GEANT4 Radiation Analysis for Space (GRAS) application is a Geant4-based tool that deals with common radiation analyses types (TID, NIEL, fluence, path length, charge deposit, dose equivalent, equivalent dose, etc.) in generic 3D geometry models.

The main requirements for a new generic tool for radiation analyses in space are ease of use, flexibility and modularity of the application. GRAS can be used for obtaining a variety of simulation output types for whichever (GDML or C++) 3D geometry model. This avoids the creation of a new tailored C++ Geant4-based application for every new project. Thanks to a modular design, the GRAS analysis type capabilities can be easily extended.

2. Tool description

User Interface (UI)

The application is controlled entirely via scripting, which can be performed at the Command:line or via macro files. All of the GRAS features are selectable via UI commands. This includes for example the selection of the geometry GDML file, the physics models, the insertion of analysis modules, and for each module the definition of the relevant parameters, such as the volumes in which the analysis has to be performed or the units for the result printout.

In the definition of the analysis to be performed during the simulation, the user can easily access the parameters of each individual module, thanks to a specific "object oriented" mechanism for the automatic creation of UI commands for each analysis module inserted. More details on the scripting and its usage can be found in the following sections: scripting examples are provided for each of the simulation components throughout the manual.

Geometry

Material manager

GRAS provides a material manager that allows the user to define arbitrary materials in run-time. This manager is a copy of the MULASSIS material manager and then, it provides the same functionalty (please refer to the MULASSIS documentation 11).

A material can be defined both,

- by its chemical formula and density
- from the NIST list of materials available in Geant4 (see Geant4 *User's Guide: For Application Developers*, Appendix 8)

Once this material is defined, it is available for the different geometry builders described below (particularly, the materials will be available for MULASSIS and Gmsh MSH geometries).

Examples of material definition

```
/geometry/material/add LiquidWater H2-O 1.0

/geometry/material/AddNIST G4_KEVLAR
```

Geometry formats

The geometry can be defined in several formats:

- a GDML text file
- a STEP file, via the commercial s/w ST-Viewer by StepTools? (http://www.steptools.com/)
- a set of macro scripting commands defining a MULASSIS-type layered geometry
- a set of macro scripting commands defining a GEMAT-type layered geometry
- an ESABASE (v.1) tessellated ES9 file
- a Gmsh MSH file (ASCII) containing a tetrahedral mesh
- a C++ class
- additionally, the geometry can be also defined by the combinantion of some of the above options, loading one or more files as geometry parts and placing/rotating them arbitrarily. Currently only GDML and Gmsh MSH geometries are supported.

The input mode for the geometry model type is selectable via UI commands at run time. If no type is specified, the C++ geometry construction class is used.

Script example:

```
/gras/geometry/type [gdml | mulassis | gemat | esabase | gmsh | multi]
```

GDML

The standard input format for the geometry is via an external file, whose format is defined by the Geometry Description Mark-up Language (GDML) 11. The input file name in which the geometry model is stored is also selectable at run time. A geometry "setup" has to be specified, as in principle a single GDML file can contain more than one set-up. The setup name is specified in the GDML file, at the end. For more information on the GDML format, please refer to the GDML documentation.

After the GDML reading, Geant4 by default strips each name from any suffix starting with 0x (such as 0x12345678). Such a suffix is often added by automated geometry format translators in order to guarantee the uniqueness of each object in the GDML XML file. One can use the UI command */gdml/setNameStripFlag [true/false]* to disable such stripping feature and retain the suffix, which can be useful for example for geometry debugging purposes.

Script example:

```
/gras/geometry/type gdml

/gdml/file satellite.gdml

/gdml/setup Default

/gdml/setNameStripFlag false
```

CAD / STEP

The user can input a geometry model described in STEP format by transforming it into tessellated format. This feature is provided via the use of a commercial S/W tool, ST-Viewer by StepTools?. In reading the model in STEP format (e.g. satellite.stp), the ST-Viewer tool generates 2 files with extensions .geom. and .tree (e.g. satellite.geom. and satellite.tree). These files contain a meshed (or tessellated) description of the geometry model, in which the shapes are described by their surfaces, and all surfaces are made of a set of facets (triangular of quadrangular). These two files can be converted to the Geant4 G4TessellatedSolid shape. GDML (from release 2.7) includes the feature to automatically read in the .geom. and .tree

files. The loaded geometry can then be output into GDML and loaded as a generic GDML model.

Script example:

```
/gdml/dumpSTViewer satellite.geom satellite.tree satellite.gdml

/gras/geometry/type gdml

/gdml/file satellite.gdml

/gdml/setup Default
```

MULASSIS

The geometry can be specified in MULASSIS format. A layered geometry can be defined using the MULASSIS script commands. The layered geometry can have a slab or sphere or cylinder shape.

```
/geometry/layer/shape slab | sphere | cylinder
```

Script example:

```
/gras/geometry/type mulassis

/geometry/layer/delete  0

/geometry/material/add Cadmium_Telluride Cd-Te    5.850E+00

/geometry/layer/shape slab

/geometry/layer/add  0 Aluminium           2    4.000E+00 mm

/geometry/layer/add  1 Cadmium_Telluride   8    0.300E+00 mm

/geometry/layer/list

/geometry/update
```

With the `update` command the MULASSIS geometry automatically sets position and direction of the source based on geometry shape and total layer thickness. The GPS centre position of the source is set at the edge of the most external layer, aimed towards the layers.

The MULASSIS geometry case uses a point source, but mimics a surface (thanks to symmetry considerations). A dedicated algorithm guarantees that the (mimicked) source surface shape and size are taken into account for the global normalisation factor:

- SLAB: source square surface area equal to the area of the slabs, with (full) side length 100x the total thickness of all layers;
- SPHERE: full sphere surface, with radius equal to the MULASSIS position of the GPS vertex, at 1.001 x the total layer thickness;
- CYLINDER: full cylinder side surface, with radius equal to the MULASSIS position of the GPS vertex, at 1.001 x the total layer thickness.

A more complete example is provided in the dedicated test available at tests/geometry/mulassis.

For more information on the MULASSIS material and geometry input format, please refer to the MULASSIS documentation 11.

GEMAT

The geometry can be specified in GEMAT format. A layered slab geometry with a user defined transverse dimension can be defined using the GEMAT script commands. Inside the layers, volumes of various shapes can be inserted.

Script example:

```
/gras/geometry/type gemat

/geometry/layer/xysize       5.60 um

#

# Define the layers

#

/geometry/layer/delete 0

/geometry/layer/add  0 Silicon 2    4.360E+00 um

/geometry/layer/list

#

# Define the depletion volumes
```

```
#

/geometry/SV/delete 0

/geometry/SV/add/boxshell     1   -1   -1.3   1.5     1.0    -0.5     -0.5
0.5  0.25    Silicon  10 um

/geometry/SV/add/boxshell     1    1    1.3   1.0     0.5     0.5      0.5
0.5  0.25    Silicon   4 um

/geometry/SV/list

#

/geometry/update
```

A complete example is provided in the dedicated test available at tests/geometry/gemat

For more information on the GEMAT material and geometry input format, please refer to the GEMAT documentation.

ESABASE

NB: This geometry input format is obsolete

The geometry can be specified in ES9 format. This approach is similar to the one described for the STEP input format. ESABASE can export the geometry as a set of meshed (or tessellated) volumes. Each volume is defined by its surfaces, and these surfaces are defined as a set of facets (triangular or quadrangular). This ESABASE tessellated export format can be converted into ES9 format by a post-processing tool.

The ES9 file contains the geometry information, and the material properties have to be defined via an additional GDML file.

This format can be particularly useful for the description of ESABASE shapes that are not supported by Geant4 (e.g. extruded shapes). Please note that information about the original shapes is lost: for example, a sphere is represented as a set of triangles, and there is no notion of radius.

Script example:

```
/gras/geometry/type esabase

/esabase/ES9FileName satellite.es9
```

```
/esabase/materialFileName materials.gdml
```

GMSH - MSH

The user can input a geometry model described in Gmsh MSH format. This format is the native meshing output from Gmsh tool (http://geuz.org/gmsh/). According the developers "Gmsh is a 3D finite element grid generator with a build-in CAD engine and post-processor. Its design goal is to provide a fast, light and user-friendly meshing tool with parametric input and advanced visualization capabilities"

GRAS supports the loading of Gmsh 3D tetrahedral meshes defined in plain-text ASCII format. Datailed information of this format cam be found in the Gmsh documentation (http://geuz.org/gmsh/doc/texinfo/gmsh.html).

Options:

- Material information: As Gmsh MSH format does not include any information about the material properties of the mesh elements, the user can explicitly assign them a Geant4 material. The material needs either to exist in the Geant4 NIST list of materials, or to have been previously defined using GRAS material manager UI commands. In the case no material is specifed, vacuum will be assigned to all mesh elements by default.
- Length unit: Coordinates/points in Gmsh MSH are dimensionless. The user can specify a Geant4 length unit to be applied to all mesh elements. If no unit is specified, it is considered that the mesh dimensions are defined in mm.
- Transfromations: Translation and rotations of the mesh as a whole are possible.

Script example

```
/gras/geometry/type gmsh

/gmsh/file myGeometry.gmsh



/gmsh/lengthUnit mm

/gmsh/material Aluminium



/gmsh/translate 10 0 15 mm

/gmsh/rotateX 45 deg
```

C++

The user can provide a C++ class with the geometry description (as in standard Geant4 applications). For help on the definition of the geometry in C++, please consult the Geant4 web pages. This mode requires the recompilation of the GRAS tool.

In case no specific geometry type is selected, and the user did not recompile GRAS with his/her own C++ geometry, GRAS provides a simple standard geometry model (hard coded in C++), loaded as a default. This model can be used for test purposes, and is available also when the GDML libraries are not available for the complete GRAS installation.

MULTI

The user can specify a geometry description by combining some of the above geometry format possibilities. Currently GRAS only supports the combination of one or more GDML and/or Gmsh MSH geometries.

The user can add an unlimited number of geometries, either GDML or Gmsh MSH, and place/rotate them arbitrarily.

Note that the user is the responsible of possible overlappings between the different loaded geometries.

```
/gras/geometry/type multi



/multi/addGDML Part01

/multi/gdml/Part01/file MyGeometry01.gdml

/multi/gdml/Part01/translate 20. 0. 0. mm

/multi/gdml/Part01/rotateY 90 deg



/multi/addGDML Part02

/multi/gdml/Part02/file MyGeometry02.gdml
```

```
/multi/gdml/Part02/translate 0. 10. 0. mm

/multi/gdml/Part02/rotateZ 45 deg



[...]



/multi/addGMSH Part03

/multi/gmsh/Part03/file MyGmshGeom.msh

/multi/gmsh/Part03/lengthUnit mm

/multi/gmsh/Part03/material Aluminium

/multi/gmsh/Part03/translate 0. 0. 0. mm

/multi/gmsh/Part03/rotateX 90 deg



[...]
```

Geometry utilities

The user can obtain information on the content of the geometry model. For the moment, the UI Command:/gras/geometry/util/listLogicalVolumes provides a list of the volumes and in addition their mass. Please note that similar information can be obtained with the ASCIITree (ATree) visualisation driver, included in Geant4.

The user can also modify the material associated to a volume by name with the Command: */gras/geometry/util/setVolumeMaterial* or */gras/geometry/util/setPhysicalVolumeMaterial*. The wildcard character '*' can be used in volume names.

Script example:

```
/gras/geometry/util/listLogicalVolumes

/gras/geometry/util/setVolumeMaterial World_log Vacuum
```

```
/gras/geometry/util/setPhysicalVolumeMaterial World Vacuum



/gras/geometry/util/setVolumeMaterial Detector* GalliumArsenide

/gras/geometry/util/setPhysicalVolumeMaterial Detector* GalliumArsenide
```

Magnetic Field Manager

A magnetic field manager is included in GRAS, allowing for introduction in the geometry model of magnetic fields with various configurations.

No documentation is included yet in this wiki, but a detailed description is provided in the separate manual

An example is also provided at trunk/gras/examples/magfield

Physics

Space radiation sources range from very low to very high energy and their interactions with the spacecraft sensitive devices and the shielding structures include both electromagnetic and hadronic processes. The GRAS tool includes therefore a very large subset of the physics models available within Geant4, with the aim of giving an almost complete coverage of the main interaction mechanisms for trapped, solar and cosmic radiation in the spacecraft materials.

GRAS offers two ways to build Physics List : in the first combining it from components (builders) using UI commands (see subsections below), in the second reference Physics List provided with Geant4 source code can be instantiated. The second variant is activated if the environment variable PHYSLIST is defined, if it is not the first variant is realised and user should use GRAS UI commands to configure Geant4 physics. In this case the user can specify his physics list as an assembly of physics modules with the Command:

```
/gras/physics/addPhysics “physics_module_of_choice”
```

A list of the available modules can be obtained with the command

```
/gras/physics/list.
```

A definition of the available modules is provided in the subsections below. A list of the inserting modules, after the assembly, can be obtained with the command

```
/gras/physics/describe
```

Electromagnetic physics

The user can select between the "standard" and "low-energy" electromagnetic packages (with the option of including the PAI model 11, of relevance for thin or low-density volumes).

Script examples:

```
/gras/physics/addPhysics em_standard | em_lowenergy
```

Hadronic physics

The user can include hadronic models for low, mid and high-energy particles. The low and mid energy hadronic models available are the Binary intra-nuclear cascade model, which describes the nuclear interactions of protons and neutrons with energy up to 10 GeV and pions up to 1 GeV, and the alternative Bertini cascade model, which describes proton, neutron and pion and kaon interactions up to 10 GeV. The Quark Gluon String (QGS) model, valid up to 100 TeV, covers higher energies and allows the simulation of a significant part of the protons in the wide Cosmic Ray spectrum.

A recent extension is also included of the Binary Cascade model for light ions with energy below 10 GeV/nucleon. However, its use for ions of higher atomic number and weight is not recommended. As an alternative, the recent Geant4 implementation 11 of the Abration/Ablation? models 1111 is available. For higher energies, an extension of the QGS model to ions 11 is under development within the Geant4 collaboration and will be included as soon as it becomes public, due to its relevance to both SEE studies and estimates of the radiation effects on manned space flight.

The Radioactive Decay Module can also be added to the physics list as an additional module

Script example:

```
/gras/physics/addPhysics elastic

/gras/physics/addPhysics binary_had
```

```
/gras/physics/addPhysics binary_ion

/gras/physics/addPhysics decay

/gras/physics/addPhysics stopping

/gras/physics/addPhysics gamma_nuc

/gras/physics/addPhysics raddecay
```

An interface to some of the Geant4 reference physics lists is also provided. In this case, only the hadronic component of the packages is used, and the user can also select the electromagnetic package (standard / low energy).

Script examples:

```
/gras/physics/addPhysics LHEP

/gras/physics/addPhysics QGSP

/gras/physics/addPhysics QGSP_BIC

/gras/physics/addPhysics QGSP_BERT

/gras/physics/addPhysics QGSP_BERT_HP
```

In addition, GRAS offers the possibility to use a GEANT4 interface to the PHITS 11 models for HZE ions 1111. Both the PHITS code and the GEANT4-PHITS interface have to be obtained privately from the respective authors.

Script example:

```
/gras/physics/addPhysics jqmd_ion
```

Control of the Reverse MC physics

A physics list has been added in GRAS for the reverse MC simulation. It takes into account at this time the following reverse and forward processes : continuous gain/loss of energy for e-, e-ionisation, e- bremstrahlung, photoelectric effect , Compton scattering. It is selected with the GRAS command

```
/gras/physics/addPhysics  rmc_em_standard
```

For testing purpose the different processes considered in this physics list can be switch on and off by using some commands in the directory /adjoint_physics/. By

default all processes are taken into account except for ion ionisation. Reverse continuous and discrete e- ionisation and protons are always considered. The minimum and maximum energies of the reverse/adjoint models can be set by the commands SetEminForAdjointModels? and SetEmaxForAdjointModels? in the directory /adjoint_physics/. For computation efficiency it is recommended to set these minimum and maximum energies to the minimum and maximum energies of the external source. Reverse ion ionisation can be taken into account for one type of ion only (proton is not considered as ion but as a separate type of particle.) by using the command /adjoint_physics/UseAdjointIon and DefineAdjointIon?. By default the ion ionisation is not considered. A list of types of adjoint primary particles that will be considered in the simulation is built during the initialisation of the physics an depends on the adjoint physics type of processes selected by the user. For example if only the ionisation for e- and proton are considered only adjoint e- and proton will be in the list of adjoint primary. While if the reverse bremsstrahlung is also selected, the adjoint gamma will also be added in the list of primaries. By default all type of adjoint primary contained in the primary list after the physics initilisation, wil be taken into account in the simulation, but the user can unselect or select separetly each of them by using the commands /adjoint/ConsiderAsPrimary and NeglectAsPrimary?.

Cuts and step limits

The user can select the Geant4 production cuts 11 and an optional limitation to the maximum step size via UI scripts.

Script example: { /gras/physics/setCuts 0.1 mm /gras/physics/stepMax 1.0 mm }

When using the Low Energy EM processes, the user can in principle set production cuts in range corresponding to cuts in energy as low as 250 eV (or even 100 eV for the Penelope models). A default lower limit of 990 eV is however used in the Geant4 kernel, and to enable lower cut values the user must change the range of the cuts with explicit instructions. GRAS offers the UI Command:/gras/physics/productionCutsEnergyRange [low] [high] for this purpose.

Script example:

```
/gras/physics/productionCutsEnergyRange 250 eV 100 GeV
```

Cuts and step limits by region

The user can define Geant4 "regions" via UI scripts, and assign cuts and step size limits by region. A region must be defined and volumes can be assigned to regions by name. Cuts and step limits can then be applied to regions.

Script example:

```
/gras/physics/regionStepMax ABSORBER 0.5 mm

/gras/physics/region/addRegion ABSORBER

/gras/physics/region/addVolumeToRegion volume_name ABSORBER

/gras/physics/region/setRegionCut ABSORBER all 0.01 mm
```

Physics utilities

The user can get some insight into the physics models via dedicated physics utilities UI commands. The code is implemented in the GRASPhysicsUtil class.

- Print the dE/dx for a set of kinetic energy points in a given range, particle type, material

```
•/gras/physics/util/printDEDXTable kinEnergyMin kinEnergyMax
  kineticEnergyUnit nPoints particle material
```

- Print the EM XS for a set of kinetic energy points in a given range, particle type, process, material [, region]

```
•/gras/physics/util/printEMCrossSectionTable kinEnergyMin
  kinEnergyMax kineticEnergyUnit nPoints particle process
  material [region]
```

- Print the EM diff XS for a given prim Energy for a set of secondary kinetic energy points in a given range, particle type, process, material [, region]

```
•/gras/physics/util/printDiffCrossSectionTable
  kinEnergyPrimary kinEnergySecondMin kinEnergySecondMax
  kineticEnergyUnit nPoints particle process material
  [region]
```

- Print the Hadronic Elastic and Inelastic XS for a set of kinetic energy points for a given target element, energy range, particle

- /gras/physics/util/printHadronicCrossSectionTable kinEnergyMin kinEnergyMax kineticEnergyUnit nPoints element particle [Z] [A]

Example:

```
/gras/physics/util/printHadronicCrossSectionTable 1
100000 MeV 51 Si proton

/gras/physics/util/printHadronicCrossSectionTable 1
100000 MeV 51 Si GenericIon 82 208

/gras/physics/util/printHadronicCrossSectionTable 1
100000 MeV 51 Si GenericIon 54 131
```

Biasing options

With the Geant4 version 9.5 and GRAS version 3.1 several biasing options become available. Electromagnetic physics can be biased using built-in Geant4 UI commands. Below an example of of UI commands is shown which includes bremsstrahlung splitting activated in a detector region "Absorber" and Russian roulette for delta electron production in the same region.

Script example:

```
/process/em/setSecBiasing eBrem  Absorber 10  5 MeV

/process/em/setSecBiasing eIoni  Absorber 0.1 2.5 MeV
```

Geometry based biasing is available for planar and spherical geometries. If geometry is describe in the MULASSIS format then this type of biasing may be enabled by simple GRAS UI commands.

Script example:

```
# must come before /geometry/update

/gras/geom_biasing/switch true

/gras/geom_biasing/particle e-
```

```
#

/geometry/update
```

Source

GRAS uses the General Particle Source (GPS) as its source generator for the primary particles. GPS offers many options for the choice of particle type, energy spectrum, emission point and emission direction distribution, with several useful predefined options and the possibility of user defined spectra. Source biasing is also part of GPS. A detailed documentation on the use of GPS can be found in 11.

Script example:

```
/gps/particle proton

/gps/ene/type Arb

/gps/hist/type arb



/gps/hist/point 1.0023 202.26

/gps/hist/point 1.0162 207.42

/gps/hist/point 1.0304 212.75

...

/gps/hist/point 97277 0.000065036

/gps/hist/point 98629 0.000062654

/gps/hist/point 100000 0.000060364

/gps/hist/inter Lin



/gps/pos/type Surface

/gps/pos/shape Sphere

/gps/pos/radius 0.5 m

/gps/pos/centre 0. 0. 0. m
```

```
/gps/ang/type cos

/gps/ang/mintheta  0 deg

/gps/ang/maxtheta 90 deg
```

Analysis

GRAS adopts a modular approach to the analysis of the impact of radiation to the systems. The user can select to perform in parallel several analysis types on independent set of volumes or volume boundaries. All analysis details can be specified via UI script commands, from the type of analysis to be performed, to the volumes to which it is applied, to the output unit and format.

More details on the analysis modules, on their usage and on the available analysis types can be found in the dedicated Section 3.

Script example:

```
/gras/analysis/dose/addModule doseDetector

/gras/analysis/dose/doseDetector/addVolume Detector

/gras/analysis/dose/doseDetector/addVolumeInterface World  Detector

/gras/analysis/dose/doseDetector/setUnit rad

/gras/analysis/dose/doseDetector/verbose 2
```

Analysis in detail

Existing analysis types include cumulative effects such as Total Ionizing Dose (TID) or Total Non Ionizing Dose (TNID), to study performance degradation in scientific detectors, commercial payload components or service elements such as solar arrays. Specific modules have been developed for the evaluation of the effects of radiation to humans: “Dose Equivalent” analysis, based on LET quality factors (QF); and “Equivalent Dose” based on the incident particle weights are available in GRAS, with a choice among several factor functions from existing protocols in the literature. Fluence of primary or secondary particles crossing a volume boundary is tallied in the Fluence modules. In addition to the total integrated fluence by particle, individual energy spectra are recorded for the particle types requested by the user. Specific modules implementing single event effects models available in the

literature have been developed. Transient effects are addressed at present by Pulse Height, included in the dose modules, an LET module and a "track length" analysis.

All analyses can be applied to any user defined set of volumes or surfaces, chosen among those present in the geometry model.

Results are saved as histograms or "tuples" (tabular data for later analysis), while basic summary quantities are also output in simple text format. Users also have the choice of the units for the result output.

Analysis module

The core part of the analysis is constituted by the "analysis modules", which can be inserted by the user via UI commands to perform the individual analysis tasks. The modular approach keeps analysis types independent and isolates each complete analysis inside one class.

After the setup of the analysis, the user can output to the terminal the main parameters of the inserted volumes with the Command:/gras/analysis/describe, with the optional verbosity integer parameter to increase/decrease the level of detail.

Script example:

```
/gras/analysis/dose/addModule doseDetector

...

/gras/analysis/describe 2
```

"Object oriented" scripting

The analysis can be performed in parallel by many modules during the same simulation, and each module, even if of the same type (e.g. "dose") can have different parameters (e.g. it can be applied to different volumes). To offer an intuitive access to all parameters of each individual module,

GRAS adopted an "object oriented" scripting interface, following the example of the GATE application 11. Each time a module is created, a new set of UI scripting commands are automatically created. These commands contain the module name in them, and are therefore unique to the module.

Script example:

```
/gras/analysis/dose/addModule doseDetector1

/gras/analysis/dose/doseDetector1/addVolume Detector1

/gras/analysis/dose/addModule doseDetector2

/gras/analysis/dose/doseDetector2/addVolume Detector2
```

Volumes and boundaries for the analysis

All analyses can be applied to any user defined set of volumes or surfaces, chosen among those present in the geometry model. Some modules perform the analysis "in the volumes", others "at the boundaries", others combine the two to get more advanced results.

The user can insert (for each module, independently) a series of volumes and/or a series of boundaries at which the relevant quantities are tallied, thus obtaining the results in complex shapes automatically (and with the related uncertainty). This avoids the need to perform several parallel analyses and combine results and uncertainties at post processing.

The wildcard symbol '*' can be used in volume names (e.g. * identifies all volumes including the "world" volume, or 'det*' all volume names beginning with 'det').

Script example:

```
/gras/analysis/dose/addModule doseDetector

/gras/analysis/dose/doseDetector/addVolume Detector1

/gras/analysis/dose/doseDetector/addVolume Detector2

/gras/analysis/dose/doseDetector/addVolume *Target*

/gras/analysis/dose/doseDetector/addVolumeInterface Box Detector1

/gras/analysis/dose/doseDetector/addVolumeInterface * Detector2
```

If necessary, in case the selection of the volumes by name is not unique, the user can specify as additional parameters, specific copy numbers in the definition of volumes and volume interfaces. In case the copy number must be ignored, the user can use the value -1: in this case, all volume copies are accepted.

Command:syntax:

```
/gras/analysis/dose/doseDetector/addVolume volume_name [copy_number]

/gras/analysis/dose/doseDetector/addVolumeInterface volume_name1
volume_name2

[copy_number1] [copy_number2]
```

Script example:

```
/gras/analysis/dose/doseDetector/addVolume Box1

/gras/analysis/dose/doseDetector/addVolume Box2 2

/gras/analysis/dose/doseDetector/addVolumeInterface * Box1 -1 1
```

Results by particle type

For many analysis types the results for can be split into contributions coming from the different "incident particle types". The definition of "incident" particle depends on the track crossing of user defined volume interfaces. A volume interface is the boundary between 2 volumes. When a track is crossing the boundary, its type is stored as "incoming particle type" and all secondary particles produced by that "incident" particle will be given the same "incident particle type", unless they pass one of the user defined boundaries themselves.

The mechanism is applied for example to all cumulative effect analyses. As an example, it is used in the "Dose Modules", to distinguish the contribution to the TID from different "incident" particle types, as opposed to different "local" types of radiation depositing a dose at a point. The usefulness of this method is apparent when analyzing shielding effectiveness from the separate effects of charged and neutral particles, as the latter only deposit energy locally through the charged products after interaction. Typical applications are the separate contributions after shielding given by input electrons and generated Bremsstrahlung photons, or by spallation neutrons produced from hadronic interactions.

Script example:

```
/gras/analysis/module_type/module_name/addVolumeInterface Volume1 Volume2
```

Uncertainties

Visualisation

As introduced in Section 2.6.2, the visualisation of the events can be triggered by the analysis modules. The UI Command `setVisTrigger` is available to set/unset this option for each module. By default the modules do not trigger the event visualisation.

Script example:

```
/gras/analysis/module_type/module_name/setVisTrigger [true | false]



/gras/event/drawEvents trigger

/vis/drawOnlyToBeKeptEvents
```

Normalisation

GRAS offers different types of normalisation of the simulation results. By default all the simulation results are simply divided by the number of primary events, but the user can also select no normalisation at all , a normalisation to compute the geometric factor (sensitive area) of the computed signals, or a normalisation to the current/flux of particles passing trough the primary source surface. The type of normalisation can be selected by the command:

```
  /gras/analysis/setNormalisationType   typeOfNorm (NONE, NONORMALIZATION,
GEOMETRIC_FACTOR, GEOFACTOR TO_SOURCE_SURFACE, TOFLUENCE,

                                                    FLUENCE, TOFLUENCE,
PEREVT, PER_NB_EVENTS, PER_NB_EVT, PEREVENT)
```

Difference between flux and current

When speaking of normalisation, it is important to remind the difference between the concept of flux and current in particle physics. The current refers to the number of particles going trough a given surface divided by the surface area, per time unit. The flux refers to the number of particle going trough a surface element perpendicular to the particle directions divided over the area of the surface element,

per time unit. The essential difference comes form the fact that in the flux concept the surface element trough which particle are counted is always perpendicular to the particle direction, while in the current concept the element surface is fixed. In the isotropic case the omnidirectional flux is 4 times the current.

Geometric factor normalisation

In this normalisation mode the results of the simulation are multiplied by the normalisation factor:

```
fN = S/4/Nb_prim
```

with *S* the area of the source surface and *Nb_prim* the number of primary particles generated from the source. In the case of an isotropic source, the user has to multiply the simulation results by the omnidirectional fluence integrated over the spectrum used for the simulation, in order to get the results normalised to the external flux. For a non isotropic case the geometric factor multiplied by 4 times the current going trough the primary source surface gives the final normalised results.

Normalisation to external flux or current

In this normalisation mode the results are normalised to a primary source fluence that is defined by the user through the command:

```
/gras/analysis/setSourceFluence fluence fluence_unit (1/cm2 1/m2 1/km2
1./cm2 1./m2  1./km2 cm-2 km-2 m-2)
```

The user can specify if the fluence is related to an isotropic omnidirectional flux or to the current going trough the primary surface. This is done by using the command

```
/gras/analysis/setSourceFluenceType (CURRENT FLUX)
```

By default the source fluence type is set to *FLUX*. In this *FLUX* case the normalisation factor considered in the simulations is given by:

```
fN = Fluence*S/4/Nb_prim
```

It is importnat to note that for the for the *FLUX* case it is assumed that the external flux is isotropic. Therefore the primary angular distribution should be set to a *cosine* law relative to the normal of the source surface. This is done by using the following *gps* commands

```
/gps/ang/type   cos

/gps/ang/mintheta 0. degree

/gps/ang/maxtheta 90. degree
```

This cosine law explains the 1/4 factor in the normalisation.

In the case that the fluence refers to a *CURRENT* the normalisation factor is given by:

```
  fN = Fluence*S/Nb_prim
```

In this case no restriction is needed considering the angular distribution, as the user defines through the current fluence the number of particle going trough the source surface per unit area.

Area of the primary source surface

For the "geometric factor" or "fluence " normalisation mode the area of the source surface has to be known to compute the normalisation factor. This area can be computed automatically by GRAS providing that the type of source selected in *gps* is either PLANAR or SURFACE. The user can decide to switch on/off this automatic computation by using the command:

```
/gras/analysis/setSourceSurfaceType AUTO/USER
```

where in the AUTO mode the area is computed automatically while in the USER mode the area is taken is defined with the command

```
/gras/analysis/setSourceSurface area area_unit (cm2 m2)
```

Special case: MULASSIS primary source surface

In case the MULASSIS-type of geometry is used, the MULASSIS geometry constructor, in addition to the layered geometry volumes definition, also pre-sets the GPS parameters for the specific selected geometry shape, to define a uniform irradiation.

The actual source position type in the MULASSIS is in fact always point-like, located at the edge of the first layer, with emission aimed into the layers. Thanks to the symmetry of the geometries this configuration mimics a uniform irradiation from the external surface of the layered structure.

Because of the point-like distribution, the area of the surface cannot be automatically obtained from GPS, and it has to be forced in GRAS based on the concept behind the individual MULASSIS cases:

- SLAB: irradiation from one side only of the slab square surface. Source surface set to the area of the square.
- SPHERE: irradiation from the surface of external spherical layer. Source set to the external area of the sphere.
- CYLINDER: irradiation from the side of the cylinder. Source set to the surface of the cylinder.

The geometry parameters needed to compute these surface areas are automatically retrieved from the MULASSIS geometry constructor.

Minor detail: the primary source vertex is not located exactly at the surface of the first layer, but rather at a small distance in front of the surface, so that the fluence of the particles entering the first layer through the first surface can be properly detected. This small correction is taken into account in the definition of the source surface area for the normalisation.

An isotropic irradiation in space can then be simulated by the GRAS user by using a cosine-law biased angular distribution.

Output units

The results are internally computed and stored in Geant4 units. However, the user can choose the unit he prefers for the result printout via a UI command. A series of appropriate units is available for each module, according to the analysis type.

Script example:

```
/gras/analysis/dose/doseDetector/setUnit rad
```

Histogramming

For the analysis modules that produce histograms and/or tuples, GRAS uses three histogram libraries options to book, fill and store the analysis output:

- Internal GRAS histogram classes (output to CSV files)
- External histogram libraries through the AIDA interface (optional)
- External ROOT histogram libraries (optional)

Depending on the available histogramming libraries, the user can have GRAS linked to both AIDA and ROOT, AIDA alone, ROOT alone and none (always keeping the internal GRAS histograms, of course).

The user can specify the name of the files used by GRAS CSV, AIDA and ROOT to store the output. The user specifies the stem of the filename (e.g. gras_histograms) with the commands: /gras/histo/fileName while the suffix of each file is automatically appended by GRAS:

- .root for ROOT files
- .csv for CSV files
- .hbook, .xml or .root for AIDA files.

Script example:

```
/gras/histo/fileName myfile
```

The user can modify some parameters of the histograms (number of bins and range) via UI commands: it is possible to select the histogram by progressive number of by name with the two UI commands /gras/histo/setHisto and /gras/histo/setHistoByName. The GRAS internal naming scheme foresees that each analysis module assigns to its histograms names composed of moduleName_histogramName (e.g. doseDetector_dose).

Script example:

```
/gras/histo/setHisto 1 100 0. 20.

/gras/histo/setHistoByName doseDetector_dose 100 0. 20.
```

Users can choose between linear (lin) and logarithmic (log) binning schemes. By default a linear binning scheme is used. The lin/log option can be specified via the two UI commands /gras/histo/setHisto and /gras/histo/setHistoByName. The log binning can only be selected for non negative bin edges.

Script example:

```
/gras/histo/setHisto 1 100 0. 20. MeV lin

/gras/histo/setHistoByName doseDetector_dose 100 0. 20. MeV log
```

In the special case of the Fluence analysis the GRAS internal naming scheme is slightly different, and names are composed of

moduleName_histogramName_particleName (e.g. fluenceDetector_fluence_proton).

Script example: /gras/histo/setHisto fluenceDetector_fluence_proton 100 0. 20.

The histogram and/or the tuple output can be enabled/disabled by module with the commands: /gras/analysis/analysis_type/module_name/bookHistos and /gras/analysis/analysis_type/module_name/bookTuples. Specific histograms containing results as a function of the primary particle (kinetic) energy are disabled by default and can be enabled/disabled by module with the command: /gras/analysis/analysis_type/module_name/bookHistosVsPrimary.

Script example:

```
/gras/analysis/dose/doseDetector/bookHistos          [true | false]

/gras/analysis/dose/doseDetector/bookHistosVsPrimary [true | false]

/gras/analysis/dose/doseDetector/bookTuples          [true | false]
```

Histograms are created by analysis modules at module initialisation phase, which is started at the latest at begin of run. Users can force earlier initialisation of single modules with the command: /gras/analysis/analysis_type/module_name/initialise. Initialisation of all inserted modules at once can be obtained with the UI Command:/gras/analysis/initialise. This enables modifications of the histogram parameters with the previously described UI commands /gras/histo/setHisto and /gras/histo/setHistoByName before the begin of run.

Internal GRAS histograms

Histograms produced by the analysis are always booked and filled with the internal GRAS histograms classes (GRASHisto1D, GRASHisto2D), which are derived from the histogram classes used in SSAT 11.

The internal GRAS histogram classes are used to generate the CSV text output, which is then always available complete with histograms, even if neither AIDA nor ROOT libraries are used. The CSV output is based on the SpenvisCSV package, separately developed to provide input/output compatible with the CSV format used inside the SPENVIS system 11.

AIDA Histograms / Tuples

Histograms and tuples produced by the analysis can also be stored with external histogram libraries through the optional “Abstract Interfaces for Data Analysis”

(AIDA). The AIDA output can be obtained only if a compatible AIDA library is installed.

The available options for the output format of the AIDA library depend on the specific AIDA concrete implementation built into GRAS (e.g. CERN/PI, OpenScientist?, JAS). The user can choose among those available formats with the Command:/gras/histo/AIDAFileType.

Script example:

```
/gras/histo/AIDAFileType [hbook | xml | root | ...]
```

ROOT Histograms/ Tuples

Histograms amd tuples produced by the analysis modules can also be saved in ROOT format via the ROOT external library 11. In ROOT histogram names the special characters '/', '+' and '-' are substituted by '_' (underscore). The ROOT output can be obtained only if a compatible ROOT installation is present.

Analysis types

Dose analysis

The dose module computes the total (cumulative) "ionizing dose" (TID) in the selected volumes. The dose module also saves the energy deposited at each event in a histogram and in a tuple, to enable the user to get the event "pulse height spectrum".

The dose is tallied in the volumes specified with the Command: `addVolume`.

Units available:

- MeV, MeV/g, rad, Gy

Where required, the required detector mass is automatically obtained from the inserted geometry volume.

If a volume interface is added with the command `addVolumeInterface`, the dose is also tallied by particle species. In this case, the contribution is subdivided into the individual contributions of the particle species as detected at the interface, i.e. not the local particle depositing the energy at the current step, but the (progenitor)

particle that crossed the user interface and finally led to the current step. This allows for example to distinguish dose from gammas / neutrons whose dose is locally deposited via their secondary particles.

If a volume interface is added, if `detailedTuples` is set to `true`, a step-by-step tuple is added with `(x, y, z)` position of the (progenitor) particle crossing. This can be useful to analyse the weak sectors in the shielding.

Script example:

```
/gras/analysis/dose/addModule dose1

/gras/analysis/dose/dose1/addVolume volume_name1

/gras/analysis/dose/dose1/addVolumeInterface volume_name2 volume_name2

/gras/analysis/dose/dose1/setUnit rad
```

Dose equivalent analysis

The dose equivalent analysis module computes the total (cumulative) "dose equivalent" in the selected volumes. The dose equivalent calculation takes into account the Relative Biological Effectiveness (RBE) of radiation as a function of particle type and energy through the use of QF. The Q(L) relationship between the QF and the LET has been implemented in GRAS based on the ICRP 60 recommendations 11.

Units available:

- Sv, mSv

Script example:

```
/gras/analysis/dose_equivalent/addModule doseEqBody

/gras/analysis/dose_equivalent/doseEqBody/addVolume Body

/gras/analysis/dose_equivalent/doseEqBody/addVolumeInterface Hall Body

/gras/analysis/dose_equivalent/doseEqBody/setUnit mSv
```

Equivalent dose / tissue effects analysis

The equivalent dose analysis module computes the total (cumulative) "equivalent dose" in the selected volumes. The equivalent dose estimate makes use of radiation Weighting Factors (wR): the user has the choice between the values adopted in ICRP 60 11, the re-appraisal of the factors given in ICRP 92 and the ones provided in the "2007 Recommendations of the ICRP" (ICRP 103, default in GRAS). The UI Command:for this selection is:

```
/gras/analysis/equivalent_dose/eqDose_module/setCurve [ICRP60 | ICRP92 | ICRP103]
```

Dose equivalent weighting factors (wR)			
Particle	**ICRP 60**	**ICRP 92**	**ICRP 103**
gamma	1	1	1
e-, e+, mu+, mu-	1	1	1
proton	5 (E>2MeV)	2 (E>10MeV)	2 (all energies)
pi+, pi-			2 (all energies)
neutron	ICRP 60 binned function	ICRP 92 function	ICRP 103 function
alpha, ion	20	20	20
others	20	20	20

Figure 1: Equivalent dose weighting factors (wF) for neutrons from ICRP 60, ICRP 92 and ICRP 103, as implemented in the GRAS equivalent dose analysis module.

When used with the relative biological effectiveness (RBE) weighting factors as recommended by ICRP draft publication July 2012, ref. 4819-7515-1888, the analysis module can also be used to obtain results for tissue effects (in units of Gy-eq). The ICRP58 weighting factor table data was taken from:

- ICRP Publication 58 (1989)
- Wilson et al., J.Radiat.Res.,43:Suppl.,S103-S106(2002)
- NCRP Publication 132

RBE weighting factors	
Particle	**ICRP 58**

neutron	5 (E<1MeV) 6 (1MeV<=E<5MeV) 3.5 (E>=5MeV)
proton	1.5 (E>2MeV)
alpha, ion	2.5
others	1

A special technique has been developed in the GRAS application to ensure the correct weighting factor is applied to the dose deposited by the particles (the primaries and the generated secondary products) according to the "incident" particle type: once a weight wR is assigned to a particle, it is then propagated to all its secondary interaction products. The definition of "incident particle" is also not restricted to the primary particle: the user can define as "incident" particles that cross a selected boundary. This is crucial for dosimetry applications in which the primary radiation interacts with a shielding structure before reaching the sensitive target: in this case there is a clear difference between the particles that are "incident" onto the target (astronauts, dosimeters, etc.), which consist of slowed-down primaries and secondary particles, and the primary particle. The user can select the boundary that defines a particle as "incident" module by module, so that analyses can be performed in parallel during the same simulation run on different sensitive target volumes, e.g. whole human body and internal organs.

Units available:

- Sv, mSv, Gy-eq

Script example:

```
/gras/analysis/equivalent_dose/addModule eqDoseBody

/gras/analysis/equivalent_dose/eqDoseBody/addVolume Body

/gras/analysis/equivalent_dose/eqDoseBody/addVolumeInterface Hall Body

/gras/analysis/equivalent_dose/eqDoseBody/setCurve ICRP103

/gras/analysis/equivalent_dose/eqDoseBody/setUnit mSv
```

Fluence analysis

The fluence analysis module computes the total (cumulative) "fluence" at the selected boundaries. The results are given as total fluence (scalar), fluence energy spectrum (histogram) and individual hits on the boundaries (tuple).

The fluence is tallied at the boundary specified with the

Command: `addVolumeInterface`, while the volumes specified with the

Command: `addVolume` are ignored.

```
/gras/analysis/fluence/addModule module_name

/gras/analysis/fluence/module_name/addVolumeInterface volume1 volume2

/gras/analysis/fluence/module_name/addVolumeInterface volume3 volume4
```

The fluence is normalised to the target volume surface area (i.e. the surface area of the 2nd volume in the inserted volume interfaces) to obtain a proper fluence calculation in units e.g. of `counts/cm2`. This area is by default automatically computed by GRAS, with the target area used for normalisation then the full target volume surface divided by 4. For the calculation of the transmitted fluence, this needs to be used in conjunction with the option `surfaceCorrection false` (see details here below).

The interface (target) surface area can also be set by the user with the command `setInterfaceSurfaceArea`, e.g. in

```
/gras/analysis/fluence/module_name/setIntefaceSurfaceArea 12 cm2
```

The automatic surface calculation can be enabled (default) / disabled with the command

```
/gras/analysis/fluence/module_name/autoInterfaceSurfaceArea [true/false]
```

The user needs to select the particle species for which a separate fluence energy spectrum must be produced (e.g. e- / proton / gamma / …) with the UI

Command: `insertParticle`. Note: ions appear each with its specific name, for

example, Al ions injected with the GPS command lines `/gps/ion 13 26` appear in

the Geant4 transport with the name `Al26[0.0]`.

Script example:

```
/gras/analysis/fluence/module_name/insertParticle gamma

/gras/analysis/fluence/module_name/insertParticle e-

/gras/analysis/fluence/module_name/insertParticle A126[0.0]
```

An extra option is available for tallying all particles, irrespective of their type/name

```
/gras/analysis/fluence/module_name/particleTallyMode [inserted | all]
```

By setting the mode to `all`, a different set of histograms are available in the output, and detailed event-by-event output is available in the fluence ROOT tuple, where particle types can be distinguished also by Z, A.

The user has the option to enable/disable a cosine correction factor to take account of the effective surface seen by the incident particle. The extra factor is a cosine of the angle with respect to the local normal to the surface. By default, the surface correction option is disabled.

```
/gras/analysis/fluence/module_name/surfaceCorrection [true | false]
```

By default, the fluence is only tallied when the particles cross the selected boundary in the direction specified by the user (i.e. the order of the volume names). The user can select to tally the fluence in both directions (NB: the fluences are added, not subtracted) with the Command: `bothWayTally`. By default, the crossing is considered only "one way", in the direction specified by the user with the Command: `addVolumeInterface` (volume1 to volume2)

```
/gras/analysis/fluence/module_name/bothWayTally [true | false]
```

Example script: a calculation of the transmitted fluence at a target volume can be obtained with the following set of UI commands to define the fluence analysis module parameters:

```
/gras/analysis/fluence/addModule fluenceLayer-2

/gras/analysis/fluence/fluenceLayer-2/addVolumeInterface Layer-1 Layer-2

/gras/analysis/fluence/fluenceLayer-2/insertParticle e-

/gras/analysis/fluence/fluenceLayer-2/insertParticle gamma
```

```
/gras/analysis/fluence/fluenceLayer-2/insertParticle proton

/gras/analysis/fluence/fluenceLayer-2/surfaceCorrection false

/gras/analysis/fluence/fluenceLayer-2/autoInterfaceSurfaceArea true

/gras/analysis/fluence/fluenceLayer-2/initialise

/gras/histo/setHistoByName  fluenceLayer-2_fluence_e-     80 1.0000e-04
1.0000e+04 MeV log

/gras/histo/setHistoByName  fluenceLayer-2_fluence_gamma  80 1.0000e-04
1.0000e+04 MeV log

/gras/histo/setHistoByName  fluenceLayer-2_fluence_proton 80 1.0000e-04
1.0000e+04 MeV log
```

NIEL analysis

Non Ionising Energy Loss (NIEL) analysis modules compute the Total Non Ionising Dose (TNID). The "NIEL" name, which refers to the non-ionising component of the rate of energy loss is incorrect but is temporarily kept for historical and macro-file backward-compatibility reasons.

The TNID is computed with a particle-by-particle convolution of the fluence with the NIEL (taken from tabulated curves, see below). The convolution is performed at the particle trajectory crossing of user selected boundaries, in a so-called "macroscopic approach". The TNID computation is therefore based on user-assigned NIEL coefficients at the volume interface (normally when entering the volume of interest) and not on the real material of the entered volume, so the user can obtain the TNID results at boundaries between volumes of arbitrary materials (provided the related NIEL tables are available for that material).

For information, a TNID calculation with a step by step calculation inside the volumes, is available in GRAS through the NID analysis type described further below.

A correction factor of 1/cos(theta) is applied, where theta is the angle between direction of the incoming particle with respect to the surface, to take account of the expected path of the particle in the target volume. This implies that the volume is assumed to be planar (because of the approximate cosine correction for the track length) and infinitesimally thin (because the particle is assumed to have constant properties, e.g. energy, inside the volume). This correction is disabled by default, but the user has the option to enable/disable the 1/cosine pathlength correction with the UI command: `pathlengthCorrection`.

Script example:

```
/gras/analysis/niel/module_name/pathlengthCorrection [true | false]
```

NIEL curve sets

The analysis module offers a choice among several NIEL coefficient tables, obtained from the literature for different semiconductor materials. The following table summarises the available curves and their applicability.

GRAS curve label	Particles	Description
`JPL_NRL_NASA_Si`	p, e-, n	JPL/NRL/NASA (2003) NIEL Curves for Si - Supplied by S. Messenger (NRL)
`JPL_NRL_NASA_GaAs`	p, e-, n	JPL/NRL/NASA (2003) NIEL Curves for `GaAs` - Supplied by S. Messenger (NRL)
`JPL_NRL_NASA_InP`	p, e-, n	JPL/NRL/NASA (2003) NIEL Curves for InP - Supplied by S. Messenger (NRL)
`GaAs`	p, e-	Data for `GaAs` from Summers. 1993
`InP`	p, e-	Data for InP from Summers. 1993
`ROSE`	p, n, pi, e-	ROSE dataset for Si from CERN RD48
`SPENVIS`	p	Data for Si supplied by Daniel Heynderickx, used in an old version of SPENVIS

Particle and energy ranges, and GRAS module behaviour:

- For particles not included in the user selected curve set
  - Particle ignored
  - With module verboseLevel > 0 a warning is printed to log
- For particle energy values
  - Within the available data set range --> the NIEL is computed by interpolation
  - Below the available range --> NIEL = 0 (NB: this assumes that data covers entirely the relevant low energy part of the NIEL curve)
  - Above the available range --> NIEL = NIEL at upper range limit

NIEL curve references:

JPL_NRL_NASA

- I. Jun, M.A. Xapsos, S.R. Messenger, E.A. Burke, R.J. Walters, G.P. Summers and T. Jordan, Proton nonionizing energy loss (NIEL) for device applications, IEEE Trans. Nucl. Sci., Vol 50, 1924-1928, 2003
- S.R. Messenger, E.A. Burke, M.A. Xapsos, G.P. Summers, R.J. Walters, I.Jun, T.M. Jordan, NIEL for heavy ions: An analytical approach, IEEE Trans. Nucl. Sci., Vol.50, 1919, 2003
- G.P. Summers, E.A. Burke, P. Shapiro , S.R. Messenger, R.J Walters, Damage Correlations in Semiconductors exposed to Gamma, Electron and Proton Radiations, IEEE Trans. Nucl. Sci., Vol. 40, No. 6., Dec. 1993
- Private communication from S. Messenger (NRL), c.f. Mott e-differential-scattering
- Standard Practise for Characterizing Neutron Energy Fluence Spectra in Terms of an Equivalent Monoenergetic Neutron Fluence for Radiation Hardness Testing of Electronics, ASTM International Standard E722-94 (Re-approved 2002), American Society for Testing Materials
- S.R. Messenger, E.A. Burke, J. Lorentzen, R.J. Walters, J.H. Warner, G.P. Summers, S.L. Murray, C.S. Murray, C.J. Crowley, N.A. Elkouh, The correlation of proton and neutron damage in photovoltaics, Proceedings 31st IEEE Photovoltaic Specialists Conference, Lake Buena Vista, Florida, Jan 3-7, 2005
- P. Griffin (Sandia Natl. Lab), Private communication

Summers

- G.P. Summers, E.A. Burke, P. Shapiro , S.R. Messenger, R.J Walters, "Damage Correlations in Semiconductors exposed to Gamma, Electron and Proton Radiations, IEEE Trans. Nucl. Sci., Vol. 40, No. 6., Dec. 1993

ROSE

- A. Vasilescu and G. Lindstroem, Displacement damage in Silicon, on-line compilation: http://sesam.desy.de/~gunnar/Si-dfuncs

SPENVIS

- JPL (?)

Units available:

- `Mev/g`, `MeV/mg`, `keV/g`, `95MeVmb/cm2`
- `10MeVp/cm2`, `50MeVp/cm2`, `1MeVe/cm2`, `1MeVn/cm2`
- `Mevcm2/g`, `MeVcm2/mg`, `keVcm2/g`, `95MeVmb`

- `10MeVp`, `50MeVp`, `1MeVe`, `1MeVn`

The first set of units require the definition of a target surface area. This area is automatically computed by the GRAS module, based on the inserted volume interface, but it can be set/overwritten by the user with the command `setInterfaceSurfaceArea`.

The units `10MeVp/cm2`, `50MeVp/cm2`, `1MeVe/cm2`, `1MeVn/cm2` (`10MeVp`, `50MeVp`, `1MeVe`, `1MeVn`) provide a normalisation of the result to an equivalent fluence(/number) of respectively 10 MeV protons, 50 MeV protons, 1 MeV electrons or 1 MeV neutrons, i.e. to the particle fluence(/number) that would produce the same TNID result. The internal conversion factors for these units are updated at each change of the NIEL curve set to use the corresponding NIEL coefficients. If the coefficients for the needed particle species are not available from the chosen NIEL curve set, the corresponding units are not available.

Script example:

```
/gras/analysis/niel/addModule nielDetector

/gras/analysis/niel/nielDetector/addVolumeInterface Hall Detector

/gras/analysis/niel/nielDetector/setCurveSet JPL_NRL_NASA_Si

/gras/analysis/niel/nielDetector/setUnit MeV/g
```

NID analysis

Non Ionising Dose (NID) analysis modules also compute the Total Non Ionising Dose (TNID).

The `NID` algorithm for TNID calculation is also based on the same NIEL curves as the `NIEL` analysis above, but in contrast to the `NIEL` analysis the `NID` calculation is performed inside the volume, step by step.

NIEL curve sets

Plese refer to the `NIEL` module documentation above for a description of the NIEL coefficient tables.

Units available

- `Mev/g`, `MeV/mg`, `keV/g`, `95MeVmb/cm2`
- `10MeVp/cm2`, `50MeVp/cm2`, `1MeVe/cm2`, `1MeVn/cm2`
- `Mevcm3/g`, `MeVcm3/mg`, `keVcm3/g`, `95MeVmbcm`
- `10MeVpcm`, `50MeVpcm`, `1MeVecm`, `1MeVncm`

The first set of units require the definition of a volume (e.g. in cm3) for the target . As done for the surface area in the fluence analysis, this volume for the NID analysis is automatically computed by the GRAS NID module, as the sum of the volumes of all inserted volumes. The automatic calculation can be enabled/disabled with the command

```
/gras/analysis/nid/module_name/autoVolume [true/false]
```

The value of the volume can be set/overwritten by the user with the command

```
/gras/analysis/nid/module_name/setVolume value unit
```

with unit in [mm3, cm3, m3].

The units `10MeVp/cm2`, `50MeVp/cm2`, `1MeVe/cm2`, `1MeVn/cm2` (`10MeVp`, `50MeVp`, `1MeVe`, `1MeVn`) provide a normalisation of the result to an equivalent fluence(/number) of respectively 10 MeV protons, 50 MeV protons, 1 MeV electrons or 1 MeV neutrons, i.e. to the particle fluence(/number) that would produce the same TNID result. The internal conversion factors for these units are updated at each change of the NIEL curve set to use the corresponding NIEL coefficients. If the coefficients for the needed particle species are not available from the chosen NIEL curve set, the corresponding units are not available.

Script example

```
/gras/analysis/nid/addModule nidDetector

/gras/analysis/nid/nidDetector/addVolume Detector

/gras/analysis/nid/nidDetector/setCurveSet JPL_NRL_NASA_Si

/gras/analysis/nid/nidDetector/setUnit MeV/g
```

Path length analysis

The path length analysis module computes the total path length in the selected volume(s). The path length is the sum of all step lengths of all particles inside the volumes.

In SEU analyses, the path length analysis can be useful for the generation of step-length histograms for later convolution with LET spectra (which can be obtained with the LET analysis module).

In micro-dometric studies, this module can be used to define an average length of radiation from a given source direction for lineal energy spectra.

The results can be split into the contributions by incident particle species by specifying additional volume boundaries.

Units available:

- km, m, cm, mm, micrometer

Script example:

```
/gras/analysis/pathLength/addModule pathLengthDetector

/gras/analysis/pathLength/pathLengthDetector/addVolume Detector

/gras/analysis/pathLength/pathLengthDetector/addVolumeInterface Hall
Detector

/gras/analysis/pathLength/pathLengthDetector/setUnit mm
```

LET analysis

The Linear Energy Transfer (LET) analysis module computes the LET spectrum at user a selected volume boundary. The LET is obtained from the Geant4

G4EMCalculator class, which computes the value of dE/dx for user specified particle type and energy in a given material.

In SEU analyses the LET spectrum can be used to obtain an estimate of the SEU rate (e.g. by integrating the LET spectrum above a given threshold).

Units available:

- MeV/cm

Script example:

```
/gras/analysis/LET/addModule module_name

/gras/analysis/LET/module_name/addVolumeInterface volume1 volume2

/gras/analysis/pathLength/pathLengthDetector/setUnit MeV/cm
```

Charging analysis

The charging analysis module computes the total charge balance passing through user defined boundaries. If a particle is passing through the surface in the direction specified by the user, the charge of the particle is added, otherwise it is subtracted.

Units available:

- e

Script example:

```
/gras/analysis/charging/addModule chargingDetector

/gras/analysis/charging/chargingDetector/addVolumeInterface Hall Detector
```

Detector analysis

The detector analysis module computes the total energy deposited per event in a series of volumes as requested by the user. The information (per volume) is output to a tuple with one row added for each event in which at least one of the volumes has a non-zero deposited energy.

Units available:

- MeV

Script example:

```
/gras/analysis/detector/addModule SiDet

/gras/analysis/detector/SiDet/addVolume PINDiode1

/gras/analysis/detector/SiDet/addVolume PINDiode2

/gras/analysis/detector/SiDet/addVolume PINDiode3
```

Typical analyses of e.g. detector response can be then performed at post-processing from the tuple output, e.g. by setting signal thresholds, adding noise, constructing coincidences / anticoincidence logics.

Source analysis

The source analysis module has been conceived for the monitoring of the primary generator, so that the user can cross check the input GPS macro input versus the position, energy and direction of the particles finally used by GRAS for the simulation.

The distributions for source kinetic energy, momentum, direction angles phi/theta and weight are saved to histograms in the CSV output, normalised to 1.

The module can also save the results to a tuple, with event-by-event information. The output file size can therefore be significant if a large size of particles is used for the simulation. The number of particles for which the information is stored can be modified with the provided UI Command:setMaxEventID. By default, source particle properties will be stored for the first 10000 particles. It is suggested to first perform a detailed monitoring of the source properties with a sufficient statistics, and subsequently limit the number of stored primary particles for production runs.

Script example:

```
/gras/analysis/source/addModule source1

/gras/analysis/source/source1/setMaxEventID 1000
```

Charge Collection Analysis (CCA)

Documentation not yet available in this wiki. Example script provided as a test at tests/analysis/chargecollection

Activation

Documentation not yet available in this wiki. Example script provided as a test at tests/analysis/activation

Reaction analysis

This analysis module allows to register the projectile and products of a set of user defined interactions in a given volume. The user selects which physical processes/interactions have to be analysed in which volumes of the geometry. All type of Geant4 processes can be considered. For each occurring reaction the projectile and the products are registered in form of tuples containing the following information:

- rnb: an integer flag giving the ordering of the selected reaction/process that occurred in the selecetd volumes since the begin of the event
- prim_sec: an integer flag defining if the particle is a projectile (1) or a product (2).
- id: particle data group (PDG) code of the particle
- evt: event ID number.
- a: ion mass number of the particle (0 if not ion or nucleon)
- z: charge number of the particle
- ekin: kinetic energy in MeV
- px, py, pz : the x, y, and z components of the particle momentum

Script example:

```
/gras/analysis/reaction/addModule raddecay

/gras/analysis/reaction/raddecay/addVolume DetBox

/gras/analysis/reaction/raddecay/addProcess RadioactiveDecay
```

Mesh scoring analysis

This analysis module allows the user to define a parallel world geometry and make use of the Geant4 scoring existing fuctionalities.

The parallel world can be defined in the following formats:

- From a GDML geometry description file.
- From a Gmsh MSH format mesh.

In order to define a mesh scoring analysis, the user must:

- Add a mesh analysis module
- As minimum an identification name for the mesh, the geometry format and the geometry file
- Optionally: additional information required by the geometry format selected and/or a set of transformations (translation, rotations)
- Initialise the analysis modules by invoking

```
•/gras/analysis/initialise
```

- Define the scoring analyses by using GRAS scoring commands (similar to Geant4 ones but placed under /gras/score/)

The script example below loads two geometries, defined in GDML and Gmsh format, and scores the energy deposited and the total charge for all the mesh elements.

Note 1: In the case of Gmsh MSH format, an additonal command is used to return the scoring results in a compatible Gmsh format (that can be post-processed afterwars using the Gmsh tool).

```
/gras/analysis/mesh/MESH2/gmsh/outputFile Mesh02.result.msh
```

Note 2: As this analysis module alows the definition of un-structured parallel scoring meshes, based on arbitray geometries, some of the scoring quantities provided by Geant4 for structutred-regular meshes (boxed and cilindrical meshes) cannot be used here. In particular, in the current version of the mesh analysis module only scalar physical quantities can be safely instatianted. Other quantities, vectorial or related to the mesh elements surface, should not be defined. Scalar quantities included in this version are:

- energyDeposit, doseDeposit
- cellCharge

Script example:

```
/gras/analysis/mesh/addModule MESH1

/gras/analysis/mesh/MESH1/type gdml

/gras/analysis/mesh/MESH1/file Mesh01.gdml

/gras/analysis/mesh/MESH1/gdml/setup Default

/gras/analysis/mesh/MESH1/translate 0 0 0 mm
```

```
/gras/analysis/mesh/MESH1/rotateX 0 deg



/gras/analysis/mesh/addModule MESH2

/gras/analysis/mesh/MESH2/type gmsh

/gras/analysis/mesh/MESH2/file Mesh02.msh

/gras/analysis/mesh/MESH2/gmsh/outputFile Mesh02.result.msh

/gras/analysis/mesh/MESH2/gmsh/lengthUnit mm

/gras/analysis/mesh/MESH1/translate 0 0 0 mm

/gras/analysis/mesh/MESH1/rotateX 0 deg



/gras/analysis/initialise



/gras/score/open MESH1

/gras/score/quantity/energyDeposit eDep MeV

/gras/score/close



/gras/score/open MESH2

/gras/score/quantity/cellCharge charge C

/gras/score/close
```

Common analysis

The common analysis module performs a basic analysis of the events by tallying the number of secondary electrons, positrons and gammas generated and the total number of steps performed by all tracks in the event. The output can be used to monitor the Geant4 tracking and the consequence of the secondary particle production cuts in electromagnetic processes.

A common analysis module is automatically inserted by GRAS as last module in the analysis chain, and its results are output at the end of the simulations, so no action is required from the user side to insert a "common analysis" module.

Visualisation

Geant4 built-in visualisation features

Geant4 itself offer a very extensive set of visualisation features, which automatically produce graphics output based on user defined criteria, filtering, etc.

This is available for the visualisation of both

- geometry model
- event trajectories

Users may choose to selectively draw part of the geometry model, or to colour the solids based on criteria such as density, etc. For trajectories, Geant4 built-in filters include selection based on touched volumes, colouring based on charge, primary energy, or other properties.

These extensive features have been introduced in Geant4 after the GRAS visualisation utilities had been introduced, so the Geant4 and GRAS capabilities are partially overlapping. However, there are a few features such as global event selection (i.e. not just for single trajectories) triggered by event properties such as event dose, that offer additional ease of use to the users. These are described here below.

We encourage the GRAS users to read the related Geant4 documentation on Visualization.

Geometry visualisation

GRAS offers utilities to interactively modify some visualisation attributes of the volumes in the geometry model, through the UI script Command:

```
/gras/vis/util/setVolumeProperty.
```

The properties that the user can modify are colour, visibility, forcewireframe, forcesolid, linewidth.

The user can assign the new property to one or more volumes at a time, selecting the volumes by (logical) name, by physical_name or by material.

Script example:

```
/gras/vis/util/addColour grey         0.5  0.5  0.5

/gras/vis/util/addColour red          1    0    0



/gras/vis/util/setVolumeProperty colour red name Detector

/gras/vis/util/setVolumeProperty colour red physical_name Absorber

/gras/vis/util/setVolumeProperty colour grey material Aluminium

/gras/vis/util/setVolumeProperty visibility false name World
```

Event visualisation

The user can decides whether to display of not the trajectories of the particles for an event based on user defined criteria. The UI Command:sets the criterion for the drawing of the events. If the "trigger" mode is selected, the event is only displayed in case at least one analysis module has triggered the visualisation during the event. For more details on the analysis triggering mechanism, see Section 3.6.

```
/gras/event/drawEvents [all | none | trigger]
```

The `trigger` option also needs the Geant4 option `/vis/drawOnlyToBeKeptEvents true`. In this case, the events are only shown if at least one analysis module has "triggered" the visualisation (e.g. the event contribution to TID/TNID is >0). The UI Command `setVisTrigger` is available for each analysis module to set/unset this option.

Script example:

```
/gras/analysis/module_type/module_name/setVisTrigger true



/gras/event/drawEvents trigger

/vis/drawOnlyToBeKeptEvents
```

Trajectory visualisation

In case the event is drawn, the Command:sets the criterion for the drawing of individual trajectories in the event. Only a simple filter based on the particle charge is implemented.

Script example:

```
/gras/event/drawTracks [all | charged | neutral]
```

Distributed parallel computing

Distributed parallel computing can be obtained by launching parallel runs on different machines. Correlations between the jobs should be avoided by forcing each job to start from different initial random seeds.

The initial seeds can be input into Geant4 via a text file. In the case of GRAS the random generator used is the "Ranecu Engine", which only needs two long integers for the engine initialisation. An example random seed file content for the Ranecu Engine is the following:

```
0

1632641235      2127009234
```

The user has to provide a set of files for the job initialisation, one for each job, and insure the given seeds lead to uncorrelated simulation events. The file is then loaded into Geant4 with the UI Command:/random/resetEngineFrom

Script example:

```
/random/resetEngineFrom seeds.rndm
```

A second option is provided in GRAS to reset the random number generator. This implementation is based on the random seed generation in the "cosmic ray charging" Geant4 advanced example 11. In this case the random seeds are automatically internally generated at the beginning of each run based on the system clock. This option can be selected with the GRAS UI Command:/gras/analysis/autoSeed.

Script example:

```
/gras/analysis/autoSeed [true | false]
```

The advantage of this option is that, by specifying /gras/analysis/autoSeed true, the user does not need to provide to each job the file with the couple of random seeds for the engine initialisation. However, it is important to notice that the level of correlation of the runs is in this case not under the control of the user, and should be verified case by case.

The seeds are by default automatically appended to the analysis output filename. This feature may be disabled with the UI

command `/gras/analysis/seedsInFileName`:

```
/gras/analysis/seedsInFileName [true | false]
```

Results and formats

Output results produced include scalars, as in the case of total dose in a volume, but also histograms, as for the LET spectrum, and tables, as for the fluence hits at a surface. Histograms and tables ("tuples") are produced through the AIDA interface.

Output formats:

- AIDA output: depending on the concrete AIDA implementation used, different output formats are available for histograms and tuples. For example, using PI-1.3.12, results can be obtained in HBOOK and XML format. The AIDA output files contain histograms and tuples.
- ROOT output : if the code has been compiled with the ROOT option, histograms and tuples are also saved in a ROOT file.
- CSV output: in addition, Comma Separated Value (CSV) text files are produced (if requested) through an internal module (SpenvisCVS). CSV files are written at a rate determined by the "print modulo" parameter. The CSV output files contain both scalar and histogram results. The format of this file follows the SPENVIS standard 11.
- LOG output: GRAS prints to the terminal part of the information on the program configuration. The level of detail of the information that is available in the terminal output largely depends on the level of verbosity set by the user. Summary results are also printed to the terminal for each analysis module at the end of each run. It is advisable to save the log information into a file for later analyses.

3. Detailed description of gras macro commands

Description of gras commands

Geometry commands

Top level control

Select geometry type*:*

```
Command:      /gras/geometry/type   <type>

Guidance:     Select the geometry construction method



Parameter:    G4String type

Candidates:   gdml, mulassis, gemat, esabase

Omittable:    False
```

Notes: 1) If mulassis type geometry is selected, the full set of MULASSIS geometry commands become available. The detailed descriptions of these commands can be found in the SUM of MULASSIS, which is available online at http://reat.space.qinetiq.com/mulassis/mulassis_files/mulassis%20sum.pdf 2) If gemat type geometry is selected, the full set of GEMAT geometry commands become available. The detailed descriptions of these commands can be found in the SUM of GEMAT, which is available online at http://reat.space.qinetiq.com/gemat/docs/GEMAT_SUM_1.2.pdf

Set the verbose level

```
Command:      /gras/geometry/util/verbose          <level>

Guidance:     Set verbose level



Parameter:    G4int level

Range:                >=0

Omittable:    False
```

describe the geometry

```
Command:      /gras/geometry/util/describe  <level>

Guidance:     describe the geometry object properties



Parameter:    G4int level

Range:                >=0

Omittable:    True

Default:      0
```

Assign a sensitive detector to a logical volume

```
Command:      /gras/geometry/util/setVolumeSD <vol>  <sd>  <bool>

Guidance:     Assignment of a sensitive detector to a logical volume

Parameter 1: logical volume name

Parameter 2: sensitive detector name

Parameter 3: true/false --> Set/Unset the sensitive Detector for the volume



Parameter:    G4String vol

Candidates:   one of the logical volume names

Omittable:    False



Parameter:    G4String sd

Candidates:   Sensitive Detector name

Omittable:    False



Parameter:    G4bool bool

Candidates:   true, false

Omittable:    False
```

List the logical volume in geometry

```
Command:      /gras/geometry/util/listLogicalVolumes

Guidance:     list of logical volumes

Parameter:    None
```

Assign a material to a logical volume

```
Command:      /gras/geometry/util/setVolumeMaterial  <vol>  <mat>

Guidance:     Assign a new material to a logical volume

Parameter 1: logical volume name

Parameter 2: material name



Parameter:    G4String vol

Candidates:   Logical volume names

Omittable:    False



Parameter:    G4String mat

Candidates:   material names

Omittable:    False
```

GDML commands

specify the gdml input geometry file

```
Command:      /gdml/file  <name>

Guidance:     Select GDML input file



Parameter:    G4String name

Omittable:    False
```

specify the gdml setup to be used

```
Command:      /gdml/setup  <setup>

Guidance:     Select geometry setup in GDML input

If none is specified the GDML processor will choose the first found



Parameter:    G4String setup

Candidates:   ???

Omittable:    True

Default value: default
```

specify the gdml Version to be used

```
Command:      /gdml/version

Guidance:     Select a particular version of a given geometry setup in GDML
input

If none is specified the GDML processor will choose the first found



Parameter:    G4String version

Candidates:   ???

Omittable:    True

Default value: default
```

specify the gdml schema location

```
Command:      /gdml/schema <location>

Guidance:     Specify the GDML schema file location



Parameter:    G4String location

Omittable:    False
```

Dump geometry in GDML format

```
Command:      /gdml/dump  <file>

Guidance:     Dump the existing geometry setup to the specified file in GDML
format



Parameter:    G4String file

Omittable:    False
```

Dump a St_Viewer geometry in GDML file format

```
Command       /gdml/dumpSTViewer  <geom.> <tree> <gdml>

Guidance :    Dump a geometry setup in ST-Viewer format (.geom, .tree) to the
specified file in GDML format



Parameter:    G4String geom

Candidate:    Name of the ST-Viewer .geom file

Omittable:    False



Parameter:    G4String tree

Candidate:    Name of the ST-Viewer .tree file

Omittable:    False



Parameter:    G4String gdml

Candidate:    Name of the GDML output file

Omittable:    False
```

Commands for adjoint simulation source definition

External source

The external source is the source that would be used as primary source for a forward simulation. It represents the surface till which an adjoint particle is tracked during a reverse simulation. The maximum energy of the external source represents the energy above which adjoint particles are killed during their tracking.

Spherical external source

```
Format:       /adjoint/DefineSphericalExtSource    <X Y Z R length_unit>

Argument:     G4String  length_unit

     G4double X, Y, Z, R

Function:     Set the external source as a spherical source of radius R centered
on the position (X, Y, Z)
```

Spherical external source placed at the centre of a physical volume

```
Format:       /adjoint/DefineSphericalExtSourceCenteredOnAVolume

<vol_name R  length_unit>

Argument:     G4String vol_name, length_unit

              G4double R

Function:     Set the external source as a spherical source of radius R with
its centre placed at the centre of the physical volume identified by the name
vol_name.
```

External source on a physical volume

```
Format:       /adjoint/DefineExtSourceOnExtSurfaceOfAVolume
     <vol_name >

Argument:     G4String vol_name
```

```
Function:     Set the external source as the external surface of the  physical volume identified by the name  vol_name.
```

Maximum energy

```
Format:       /adjoint/SetExtSourceEmax  <Emax energy_unit>

Argument:     G4String  energy_unit

              G4double Emax

Function:     Set the maximum energy of the spectrum of the external source. Adjoint particles that exceed this energy are killed during the reverse tracking phase.
```

Definition of the adjoint source

The adjoint source represents the primary source from which coupled adjoint and forward particles are generated in the reverse simulation. This source is generated on a surface that can be defined by the user and should enclose the part of the geometry where detection signals will be computed. For computing efficiency it should be the smallest as possible. It can be set either as a the external surface of a sphere or of a physical volume of the geometry. The minimum and maximum energies of the adjoint source should be set by the user as the minimum and maximum energies of the real external source. A list of types of adjoint primary particles considered in the simulation is built during the initialisation of the physics an depends on the type of reverse processes selected by the user. For example if only the ionisation for e- and protons ionisation are considered only adjoint e- and protons will appear in the list of adjoint primary after the initialisationof the physics. While if the reverse bremsstrahlung is also selected, the adjoint gamma will also be added in the list of primaries. By default all type of adjoint particles contained in the list of adjoint primaries after the physics initilisation wil be taken into account in the simulation, but the user can unselect/select seperatly each of them by using the commands /adjoint/NeglectAsPrimary and ConsiderAsPrimary?. For the simulation with ions the name of the primary ion in the list of adjoint primaries is refered by the string “ion”.

Spherical adjoint source

```
Format:       /adjoint/DefineSphericalAdjSource            <X Y Z R length_unit>

Argument:     G4String  length_unit
```

```
     G4double X, Y, Z, R

Function:     Set the adjoint source as a spherical source of radius R centered
on the position (X, Y, Z)
```

Spherical adjoint source placed at the center of a physical volume

```
Format:       /adjoint/DefineSphericalAdjSourceCenteredOnAVolume
     <vol_name R  length_unit>

Argument:     G4String vol_name, length_unit

              G4double R

Function:     Set the adjoint source as a spherical source of radius R with
its center placed at the center of the physical volume

identified by the name  vol_name.
```

Adjoint source on a physical volume

```
Format:       /adjoint/DefineAdjSourceOnExtSurfaceOfAVolume <vol_name >

Argument:     G4String vol_name

Function:     Set the adjoint source as the external surface of the  physical
volume identified by the name  vol_name.
```

Minimum energy

```
Format:       /adjoint/SetAdjSourceEmin  <Emin energy_unit>

Argument:     G4String  energy_unit

              G4double Emin

Function:     Set the minimum energy of the spectrum of the adjoint primary
source.
```

Maximum energy

```
Format:       /adjoint/SetAdjSourceEmax  <Emax energy_unit>
```

```
Argument:     G4String  energy_unit

              G4double Emax

Function:     Set the maximum energy of the spectrum of the adjoint primary
source.
```

Neglect some adjoint primary type

```
Format:       /adjoint/NeglectAsPrimary  <particle_name>

Argument:     G4String  particle_name

Candidates:   Depends on the reverse physics selected. At most [e-, gamma,
proton, ion]

Function:     Neglect a given type of adjoint particle in the list of adjoint
primaries. By default all type of primaries, that should be part of the
simulation  following the user selected list of reverse processes, are
considered. This command allows to remove some of them.
```

Consider a given type of adjoint primary

```
Format:       /adjoint/ConsiderAsPrimary <particle_name>

Argument:     G4String  particle_name

Candidates:   Depends on the reverse physics selected. At most [e-, gamma,
proton, ion]

Function:     Consider a given type of adjoint particle in the list of adjoint
primaries. By default all type  of primaries that should be part of the
simulation  following the user slected list of reverse processes, are
considered.
```

Physics commands

Physics list

Add a physics list or component

```
Command:      /gras/physics/addPhysics  <phys>

Guidance:     Add predefine Physics List or its component

                        (list Command:shows all options)

Remark: PhysicsList can be combined from components

    EM physics + Hadron Elastic + Hadron Inelastic + Extra



Parameter:    G4String  phys

Omittable:    False

Candidates:

 * Predefined PhysicsList available for AddPhysics

     <LHEP>  <QGSP_BERT>  <QGSP_BIC>  <QGSP_BERT_HP> <QGSP_BIC_HP>

     <QGSC>  <QBBC>  <QBBC_HP>

 * List of alternative EM builders

    <em_standard>  <em_standard52>  <em_standard_fast>  <em_standard_exp>
<em_lowenergy>   <rmc_em_standard>

 * List of alternative builders for the Elastic Process

    <elastic>  <HElastic> <QElastic> <LElastic>

 * List of alternative builders for the Hadron Inelastic

    <binary>  <binary_had>  <binary_hp> <bertini> <bertini_hp>

 * List of alternative builders for the Ion Inelastic

    <binary_ion>  <Abrasion>  <Abrasion+Ablation>
```

```
* List of Extra components availbale for AddPhysics

   <decay> <raddecay> <gamma_nuc> <stopping>
```

4. Tutorial examples

GRAS Examples

Different GRAS examples are provided in the directory gras/examples. These examples are run by GRAS is called as a typical Geant4 application by typing

gras gras_example_macro_file_name

from the example directory.

Examples gras/examples/gras_in_external_application

The two examples N01_with_GRAS and NO4_with_GRAS located in the directory examples/gras_in_external_application illustrate how existing Geant4 applications can be modified to be linked with and use the GRAS library (analysis modules, physics list, primary generator). These examples are extensions of the novice examples N01 and N04 distributed within Geant4. The modifications of the initial novice examples consists into:

- Replacing the creation and use of G4RunManager by GRASRunManager in the main
- Removing the declaration of the user physics list and primary generation action to the run manager in the main. The physics list and primary generator action provided with GRAS will be used.
- Modification of the CMakeLists.txt file provided with the example in order to compile correctly the example codes and link their executable with the gras library.

The following step should be done in order to compile an example :

- source the GRAS_INSTALL_PREFIX/bin/gras-env.sh(csh) file (if not already done in bash(csh) configure scripts)
- create an example_build directory where to build the example
- from this directory type
    - cmake path_to_the_example_directory
    - make

After the compilation the executable of the code is located in the example_build directory.

Examples gras/examples/magfield

The two examples example1 and example2 located in the directory examples/magfield illustrate the use of the magnetic field model manager that allows to define a magnetic field with interactive macros commands. Use of the examples and of the magnetic field model manager is described in the document gras/examples/magfield/doc/user_manual_bfield_manager.pdf.

Examples gras/examples/mulassis_geometry_biasing

Examples gras/examples/normalisation/automatic_normalisation

This example presents the different automatic normalisation modes available in GRAS. Thsse normalisation modes are described under the section normalisation of the user guide.

Run of the example in the no normalisation mode

In this mode no normalisation is considered. This run mode is executed by typing:

```
gras nonormalisation.g4mac
```

The results of the run are saved in the file nonormalisation.csv.

Run of the example in the per event normalisation mode

This run mode is executed by typing:

```
gras perevent_normalisation.g4mac
```

The results of the run are saved in the file perevent_normalisation.csv.

Run of the example in the geometric factor normalisation mode

This run mode is executed by typing:

```
gras geofactor_normalisation.g4mac
```

The results of the run are saved in the file geofactor_normalisation.csv.

Run of the example in the to fluence normalisation mode

This run mode is executed by typing:

```
gras tofluence_normalisation.g4mac
```

The results of the run are saved in the file tofluence_normalisation.csv.

Example gras/examples/radioprotection/ICRU_sphere

Example gras/examples/reverse_mc/simple_geometry ¶

5. GRAS Testing Suite based on PYTHON

GRAS testing suite based on PYTHON

All the code contained in the gras/python directory represents a PYTHON test suite to launch the GRAS test and examples and look at and/or plot the simulation results saved in csv files.

Directory structure

- python_utilities: different python classes and modules:
  - start/control simulation
  - Look at the simulation results
  - python interface to spenvis (spenvis_csv subdirectory)
  - some geometry/gdml classes allowing the building of GDML file from python functions
- pyhon_scripts: contains python scripts that the user can edit/modify to use the python test suite
- csv_files: some Spenvis csv file examples
- data: contains a list of isotope from NIST, used by the MaterialManager?

- database: will contain after the first simulations a runtest Sqlite database that is used to store

running and waiting_for_run simulation.

Dependencies

The following packages should be installed

- python2.xxx (>2.5), (+library and include files)
- the python numpy library
- the mathplotlib/pylab python library for plotting of the results
- the BOOST python library needed to build the python spenvis_csv interface (see below)

Installation of the PYTHON / SpenvisCSV interface

Once the dependency packages have been installed, it is necessary to build the SpenvisCSV python interface that allows to read Spenvis csv files in python. The C++ source files representing the interface are contained in the directory python_utilities/spenvis_csv. It consists into a slight modified version of the Spenvis CSV C++ code developed by H. Evans (ESA/ESTEC) plus a pySpenvis.cc code that realizes the interface withPYTHON, by using the BOOSTPYTHON library. This SpenvisPYTHON interface can be compiled and installed by running the python script install_pyspenvis.py in the gras/python directory. THis script uses first cmake to produce th e Makefile for the compilation of Spenvis C++ code. The use of cmake allows installation of thePYTHON Spenvis interface on different types of platform with exactly the same installation procedure. At the moment the installation have been tested on Linux Suse11.3 (32/64bit) and MacOS 10.3 (64 bit).

Use of the testing suite

Run of tests

The PYTHON script python/python_scripts/perform_gras_distribution_test.py illustrates how to run some or all tests and examples of the gras distributions. At the moment the tests are run on the local directory of the test. The PYTHON test suite will be further developed to able to run all tests on a separate directory in order to keep track of previous tests results. While several tests can be chosen to be performed in the same script/call by the user, all the selected tests are not run in the same time but in sequence using a queuing system. The user can define the number maximal of test to be run in parallel (depends on the number of cores available on

the test machine). A sql_lite database is used to check that he number of running tests is lower or equal to the user defined max number of running tests. "Still to be run" tests are set in a queue, waiting for the running tests to be finished. Any time the run of a test is finished the top test in the queue system is launched for running. At the end of the run of a test it is checked if an error file has been produced and in the posotive case its content is printed on the screen.

Check of test results

The PYTHON script python/python_scripts/check_gras_distribution_test_results.py illustrates how to check the results of some or all tests and examples of the gras distributions. For all user selected tests the following is done:

- The content of the log file is printed on the screen
- The content of the error file is printed on the screen
- If the variable "view_plot" is set to True in the script check_gras_distribution_test_results all 1D Histogram registered in the GRAS csv file are plotted one after the other
- Finally the value and estimated error of all GRASdouble contained in the GRAS csv file output are printed