



**Joint ICTP-IAEA Workshop on Monte Carlo Radiation Transport
and Associated Data Needs for Medical Applications**

28 October – 8 November 2024

ICTP, Trieste, Italy

Lecture 25

egs++ applications

Reid Townson

Metrology Research Centre
National Research Council Canada



Government
of Canada

Gouvernement
du Canada



All egs++ applications require an .egsinp file

All the egs++ applications distributed with EGSnrc rely on input blocks saved in a text file ending in `.egsinp`

The input file must reside inside the same directory as the application source code, e.g., inside the `$EGS_HOME/egs_chamber` directory for the `egs_chamber` application.

Standard egs++ applications require:

1. A geometry definition
2. A particle source
3. Monte Carlo transport parameters
4. A run control input
5. Random number generator seeds
6. **Application-specific scoring input**

1. Geometry definition input

Usually, several geometries are defined and are then combined with composite geometry objects to build the final, more complex geometry. The typical layout is:

```
:start geometry definition:
```

```
  :start geometry:
```

```
    name = foo
```

```
    (...)
```

```
  :stop geometry:
```

1. Geometry definition input

Usually, several geometries are defined and are then combined with composite geometry objects to build the final, more complex geometry. The typical layout is:

```
:start geometry definition:
```

```
    :start geometry:
```

```
        name = foo
```

```
        (...)
```

```
    :stop geometry:
```

```
    :start geometry:
```

```
        name = bar
```

```
        (...)
```

```
    :stop geometry:
```

1. Geometry definition input

Usually, several geometries are defined and are then combined with composite geometry objects to build the final, more complex geometry. The typical layout is:

```
:start geometry definition:

    :start geometry:
        name = foo
        (...)
    :stop geometry:

    :start geometry:
        name = bar
        (...)
    :stop geometry:

    simulation geometry = foo    # or bar

:stop geometry definition:
```

1. Geometry definition input

Usually, several geometries are defined and are then combined with composite geometry objects to build the final, more complex geometry. The typical layout is:

```
:start geometry definition:

    :start geometry:
        name = foo
    :stop geometry:

    :start geometry:
        name = bar
    :stop geometry:

    simulation geometry = foo    # or bar

:stop geometry definition:
```

The `simulation geometry` key specifies the geometry to load in `egs_view`, but the scoring input might override this for the actual calculation geometry.

2. Source definition input

Applications get particles from the `getNextParticle()` method of the source object. If particles are created outside of the geometry, they take a single long step along their initial direction until they hit the geometry.

```
:start source definition:
```

```
  :start source:  
    name = foo  
    (...)  
:stop source:
```

2. Source definition input

Applications get particles from the `getNextParticle()` method of the source object. If particles are created outside of the geometry, they take a single long step along their initial direction until they hit the geometry.

```
:start source definition:
```

```
  :start source:  
    name = foo  
    (...)
```

```
:stop source:
```

```
  :start source:  
    name = bar  
    (...)
```

```
:stop source:
```


2. Source definition input

Applications get particles from the `getNextParticle()` method of the source object. If particles are created outside of the geometry, they take a single long step along their initial direction until they hit the geometry.

```
:start source definition:

    :start source:
        name = foo
        (...)
    :stop source:

    :start source:
        name = bar
        (...)
    :stop source:

    simulation source = foo    # or bar

:stop source definition:
```

2. Source definition input

Applications get particles from the `getNextParticle()` method of the source object. If particles are created outside of the geometry, they take a single long step along their initial direction until they hit the geometry.

```
:start source definition:

    :start source:
        name = foo
    :stop source:

    :start source:
        name = bar
    :stop source:

simulation source = foo    # or bar

:stop source definition:
```

Particles might miss the geometry! Make sure that your source and geometry are defined so that particles are inside the geometry or aimed towards it.

3. Monte Carlo transport parameters input

Monte Carlo transport parameter inputs are common to all EGSnrc applications. Default values are set to provide accurate simulation of coupled electron-photon transport.

For example:

```
:start MC transport parameter:
  Global ECUT                = 0.521      # electron cutoff (MeV)
  Global PCUT                = 0.010      # photon cutoff (MeV)

  Spin effects                = On         # [On], Off
  Brems cross sections        = NRC        # [BH], NIST, NRC
  Bound Compton scattering    = On         # [On], Off, norej
  Rayleigh scattering         = On         # [On], Off, custom
  Atomic relaxations          = On         # [On], Off
  Brems angular sampling      = KM         # Simple, [KM]
  Pair angular sampling       = KM         # Off, [Simple], KM
  Photoelectron angular sampling = On      # [On], Off
  Electron Impact Ionization = Off       # On, [Off], ...
  Photon cross sections       = xcom       # [xcom], epdl, si
:stop MC transport parameter:
```

4. Run control input

Simulations are split into **chunks** (just one chunk in serial execution) and chunks are further divided in **batches** to help in displaying progress and saving intermediate results.

The simulation is controlled by a **run control object** (RCO), which:

- reads the number of histories requested
- reports the progress of the simulation after each batch
- defines the type of simulation (first, restart, combine or analyze)
- terminates the simulation if the sought accuracy is attained
- terminates the simulation if the maximum allotted CPU time is reached.

```
:start run control:
  ncase          = 1000      # number of histories to run
  calculation     = first    # [first], restart, combine, analyze
  statistical accuracy sought = 1    # in percent (%)
  nbatch          = 10       # number of batches (default is 10)
  nchunks         = 10       # number of chunks (default is 10)
:stop run control:
```

5. Random number generator seeds

Statistically independent simulation runs require independent random number generator seeds. In egs++ applications the seeds are set via a **rng definition** input block:

```
:start rng definition:  
  # any two integers less than 30000  
  initial seeds = 91 2556  
:stop rng definition:
```

In **parallel runs**, the application object takes care of incrementing the seed so that each job in the parallel run is statistically independent.

EGSnrc bundles a few egs++ applications

The EGSnrc distribution contains some ready-made egs++ applications geared towards specific radiation transport scenarios. These applications are derived from either [EGS_SimpleApplication](#) or [EGS_AdvancedApplication](#) and are normally installed in corresponding directories under `$EGS_HOME/`.

- [tutor2pp](#), [tutor4pp](#), [tutor7pp](#): tutorial egs++ applications
- [cavity](#): ion chamber dose calculations
- [egs_chamber](#): efficient in-phantom ion chamber calculations
- [egs_fac](#): free-air chamber correction factors calculations
- [egs_cbct](#): cone-beam CT scatter correction calculations
- [egs_kerma](#): efficient kerma calculations
- [egs_gammaspec](#): detector efficiencies and coincidence summing corrections

Create **myapp** applications in \$EGS_HOME

```
$ cd $EGS_HOME  
$ mkdir myapp  
$ ls
```

```
bin/          dosrznrc/      egs_fac/       ranmar_test/   tutor3/        tutor7pp/  
beamnrc/      dosxyznrc/     examin/        sprrznrc/      tutor4/        peps4/  
cavity/       edknrc/        flurznrc/      tutor1/        tutor5/        myapp/  
cavrznrc/     egs_cbct/      g/             tutor2/        tutor6/  
cavsphnrc/    egs_chamber/   ranlux_test/   tutor2pp/      tutor7/
```

Create **myapp** applications in \$EGS_HOME

```
$ cd $EGS_HOME  
$ mkdir myapp  
$ ls
```

```
bin/          dosrznrc/      egs_fac/       ranmar_test/   tutor3/   tutor7pp/  
beamnrc/      dosxyznrc/     examin/        sprrznrc/      tutor4/   pegs4/  
cavity/       edknrc/        flurznrc/      tutor1/        tutor5/   myapp/  
cavrznrc/     egs_cbct/      g/             tutor2/        tutor6/  
cavsphnrc/    egs_chamber/   ranlux_test/   tutor2pp/      tutor7/
```

You must create the following files inside the **myapp directory, or copy them from another application (and edit the **Makefile**):**

```
Makefile  
array_sizes.h  
myapp.cpp  
myapp.macros
```


The world's smallest EGSnrc application

myapp.cpp

```
#include "egs_advanced_application.h"  
APP_MAIN (EGS_AdvancedApplication); // short-hand #define
```

The world's smallest EGSnrc application

myapp.cpp

```
#include "egs_advanced_application.h"

int main (int argc, char **argv) {

    EGS_AdvancedApplication app(argc,argv);

    // init (read input, setup data, etc.)
    int err = app.initSimulation();
    if (err) return err;

    // start (shower loop: get next particle, transport)
    err = app.runSimulation();
    if (err < 0) return err;

    // finish (print results, tidy up, etc.)
    return app.finishSimulation();
}
```

Derive your own application class

myapp.cpp

```
#include "egs_advanced_application.h"
#include "egs_interface2.h"

class APP_EXPORT my_App : public EGS_AdvancedApplication {
public:
    my_App(int argc, char **argv) : EGS_AdvancedApplication(argc,argv) {}
    int ausgab(int iarg);
};

APP_MAIN (my_App);
```

Get something out of it: **ausgab**

myapp.cpp

```
#include "egs_advanced_application.h"
#include "egs_interface2.h"

class APP_EXPORT my_App : public EGS_AdvancedApplication {
public:
    my_App(int argc, char **argv) : EGS_AdvancedApplication(argc,argv) {}
    int ausgab(int iarg);
};

// ausgab
int my_App::ausgab (int iarg) {

    // Current particle and region indices
    int np = the_stack->np - 1;           // -1 offset
    int ir = the_stack->ir[np]-2;         // -2 offset

}

APP_MAIN (my_App);
```

Get something out of it: **ausgab**

myapp.cpp

```
int my_App::ausgab (int iarg) {

    // Current particle and region indices
    int np = the_stack->np - 1;           // -1 offset
    int ir = the_stack->ir[np]-2;         // -2 offset

    // List deposited energy in region 1
    // By default, ausgab is called for iarg<5
    // which catches all energy depositions
    // (see tutor4pp.cpp for ideas...)
    if (ir == 1) { // Only if we're in region 1
        // Energy deposited = final energy * weight
        double edep = the_stack->E[np] * the_stack->wt[np];
        egsInformation("%g\n", edep); // Print it out
    }
}
```

Get something out of it: **ausgab**

myapp.cpp

```
int  tutor4_Application::ausgab (int iarg) {  
  
    // All of the stack quantities  
    int      np = the_stack->np - 1;  
    int      ir = the_stack->ir[np]-2;  
    int      iq = the_stack->iq[np];  
    double   E  = the_stack->E[np];  
    double   x  = the_stack->x[np];  
    double   y  = the_stack->y[np];  
    double   z  = the_stack->z[np];  
    double   u  = the_stack->u[np];  
    double   v  = the_stack->v[np];  
    double   w  = the_stack->w[np];  
    double   wt = the_stack->wt[np];  
    int      lt = the_stack->latch[np];  
    int      npold = the_stack->npold - 1;  
}
```

You can check for before and after many conditions

myapp.cpp

```
// ...

// For the full list of options, see egs_application.h and pirs-701
switch (iarg) {
case BeforeTransport:    echo = false;                                break;
case EgsCut:             str = "Energy_below_Ecut_or_Pcut";          break;
case PegsCut:            str = "Energy_below_AE_or_AP";              break;
case UserDiscard:        str = "User_discard";                      break;
case ExtraEnergy:        str = "Extra_Energy_deposited";             break;
case AfterTransport:     echo = false;                                break;
case BeforeBrems:        str = "Bremsstrahlung_about_to_occur";      break;
case AfterBrems:         echo = false;                                break;
case BeforePair:         str = "Pair_production_about_to_occur";     break;
case AfterPair:          echo = false;                                break;
case BeforeCompton:      str = "Compton_scattering_about_to_occur";  break;
case AfterCompton:       echo = false;                                break;
case BeforePhoto:        str = "Photoelectric_effect_about_to_occur"; break;
case AfterPhoto:         echo = false;                                break;
case BeforeRayleigh:     str = "Rayleigh_scattering_about_to_occur"; break;
case AfterRayleigh:      echo = false;                                break;

// ...
```