User routines

How to tailor your needs to the FLUKA environment

Outline

- Introduction to user routines
    - Motivations
    - How to compile and link to FLUKA
    - Most important commons

- User routine usage
    - Classification of user routines
    - usrglo, usrini: user-defined initialization
    - usrmed & MAT-PROP: user defined medium properties
    - stupre/strprf: intercepting the particle stack
    - comscw, fluscw: weighting scored quantities (energy, fluences…)
    - usrrnc: residual nuclei scoring

Why User Routines?

FLUKA naturally provides plenty of “cards” to simulate most conceivable physics cases, **without a single line of code**. However, there are a few exceptions:

- You want to simulate a specific scenario, which doesn’t fit well with the basic FLUKA usage
- You need to extract information that are not provided naturally by scoring cards

Express your needs in dedicated FORTRAN

User Routines

How to compile

- Flair natively offers a compiler, as introduced in the FLUKA environment lecture.
- This is not the only possibility, the user can also:
  - Link and compile via terminal
  - Write a makefile

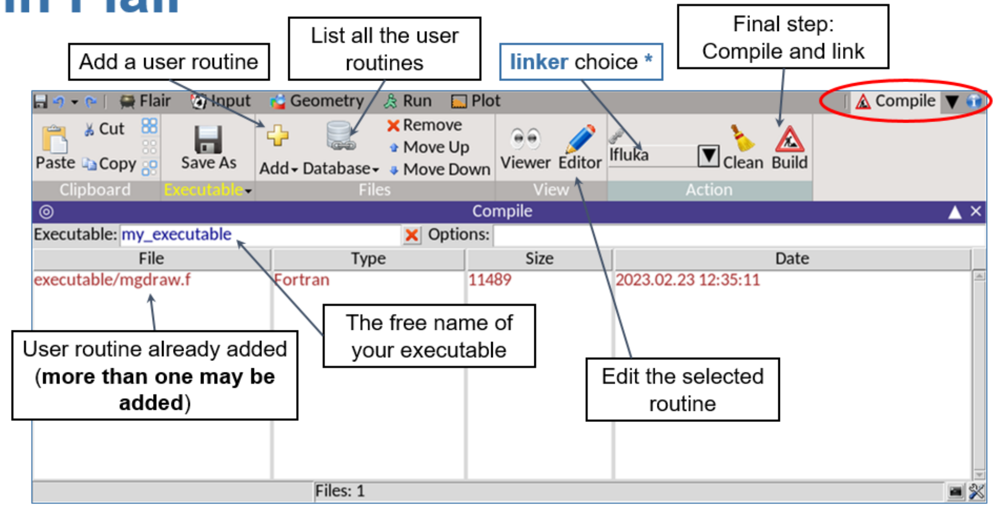
From the FLUKA environment lecture

The Compile tab in Flair

Most user routines need to be activated by input cards.

The Database button includes the Scan Input capability that automatically highlights the user routines implied by your input file.

A single executable shall embed all user routines of your choice.



Executable: my_executable   Options:

File	Type	Size	Date
executable/mgdraw.f	Fortran	11489	2023.02.23 12:35:11

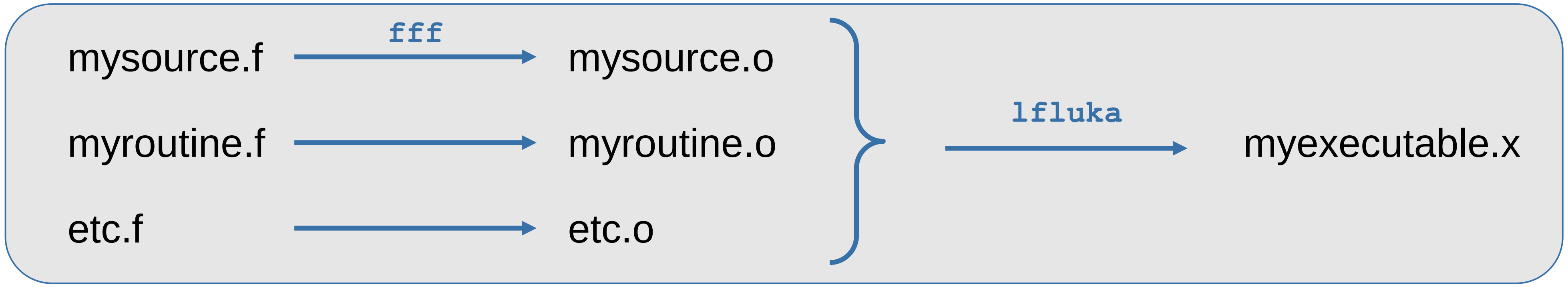
Files: 1

** Note the linker alternative **ldpmqmd**, which is required to include the DPMJET and RQMD libraries (for ions above 125 MeV/n and hadrons above 20 TeV).*

What is available for users

- Once the user routine has been written or modified, the user needs to:
  a. Compile each source routine into an object file: `/usr/local/fluka/bin/fff`
  b. Link each object file to the fluka executable*: `/usr/local/fluka/bin/lfluka`

- As good practice, try to keep everything in your working directory



Do everything outside the installation folder

```
fff xxx.f
fff yyy.f
lfluka -o myfluka xxx.o yyy.o
```

* For simulations requiring the DPMJET and RQMD packages, add the -d option, or use *ldpmqmd* instead of *lfluka*

A possible makefile

```
FLUKA=/usr/local/fluka/                                      # my installation path
FFF=$(FLUKA)/bin/fff                                         # compiler
LFLUKA=$(FLUKA)/bin/lfluka                                   # linker, I don’t need DPMJET
# LFLUKA=$(FLUKA)/bin/ldpmqmd
#
SRCFILES := $(wildcard ./*.f)                                # source files are in the same folder as the makefile
OBJECTS := $(patsubst %.f, %.o, $(SRCFILES))                 # objects have the same name, but a .o extension
PROGRAM=myexecutable.x                                       # name of the executable
#
### RULES
#
.f.o:                                                        # compile the source for each object missing
         $(FFF) $<
#
### TARGETS
#
all: $(PROGRAM)
#
$(PROGRAM): $(OBJECTS)                                       # link the objects to generate an executable
         echo $(OBJECTS)
         $(LFLUKA) -m fluka -o $@ $^
#
clean:
         rm -f $(PROGRAM) *.o *.map *.FOR
```

User routines zoology + **mgdraw.f**

User global settings

usrglo.f

User run control

usrini.f usrein.f
usrout.f usreou.f
ftelos.f

Medium properties

magfld.f
usrmed.f
elefld.f
usrhsc.f

Event generation, physics, kinematics

source.f soevsv.f
udcdrl.f formfu.f
lppsok.f ustckv.f

FLUKA output

comscw.f **fluscw.f**
endscp.f fldscp.f
musrbr.f lusrbl.f
fusrbv.f **usrrnc.f**

Particle stack interception

mdstck.f **stupre.f**
stuprf.f pshckp.f

Optical photons

abscff.f dffcff.f
frghns.f ophbdx.f
queffc.f rflctv.f
rfrndx.f wvlnsh.f

Biasing

ubsset.f
usimbs.f

Do not lose the overview!

Lots of routines, we will explore the usage of a selected few

User-defined initialisation

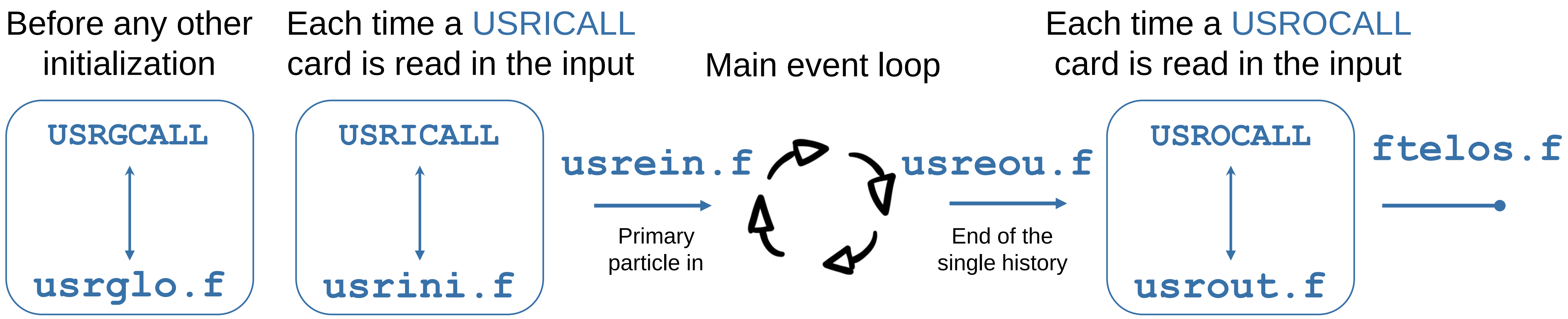
User global settings

`usrglo.f`

User run control

`usrini.f usrein.f`
`usrout.f usreou.f`
`ftelos.f`

- These routines are called at various moment during the simulation
- Some routines are activated by a corresponding card in FLUKA
- They can be used to perform any possible action outside of the event loop



Before any other initialization
Each time a USRICALL card is read in the input
Main event loop
Each time a USROCALL card is read in the input
USRGCALL
usrglo.f
USRICALL
usrini.f
usrein.f
Primary particle in
usreou.f
End of the single history
USROCALL
usrout.f
ftelos.f

User-defined initialisation: example(s)

`usrglo.f`

We want to supply another routine with numerical input parameters without recompiling the executable.

The values can be shared via appropriate **commons**

Card in the input file:

```
*...+....1....+....2....+....3..
USRGCALL          123.       456.
```

Variable assignment in `usrglo.f`:

```
MYVARA = WHAT(1)
MYVARB = WHAT(2)
```

`ftelos.f`

At the end of the simulation, we want to save some information in a dedicated file.

Normally, plenty of information is already present in the .out output file

```
OPEN ( UNIT = 99, FILE = FILNAM, STATUS = 'OLD', FORM =
     &           'FORMATTED' )

100    FORMAT (I20,3X,A)
110    FORMAT (ES20.7,3X,A)
       WRITE (99,*) "#####"
       WRITE (99,100) TPMEAN, "s -> Mean CPU time per primary"
       WRITE (99,110) NCASE,  "  -> Total number of primaries"
```

Why?

Quality of life improvement!

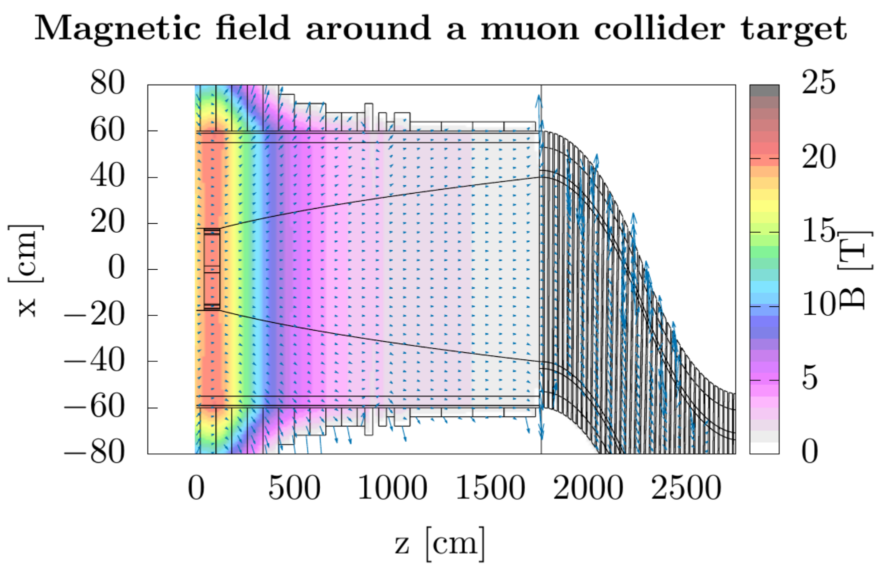
User-defined medium properties

Medium properties

`magfld.f`
`usrmed.f`
`elefld.f`
`usrhsc.f`

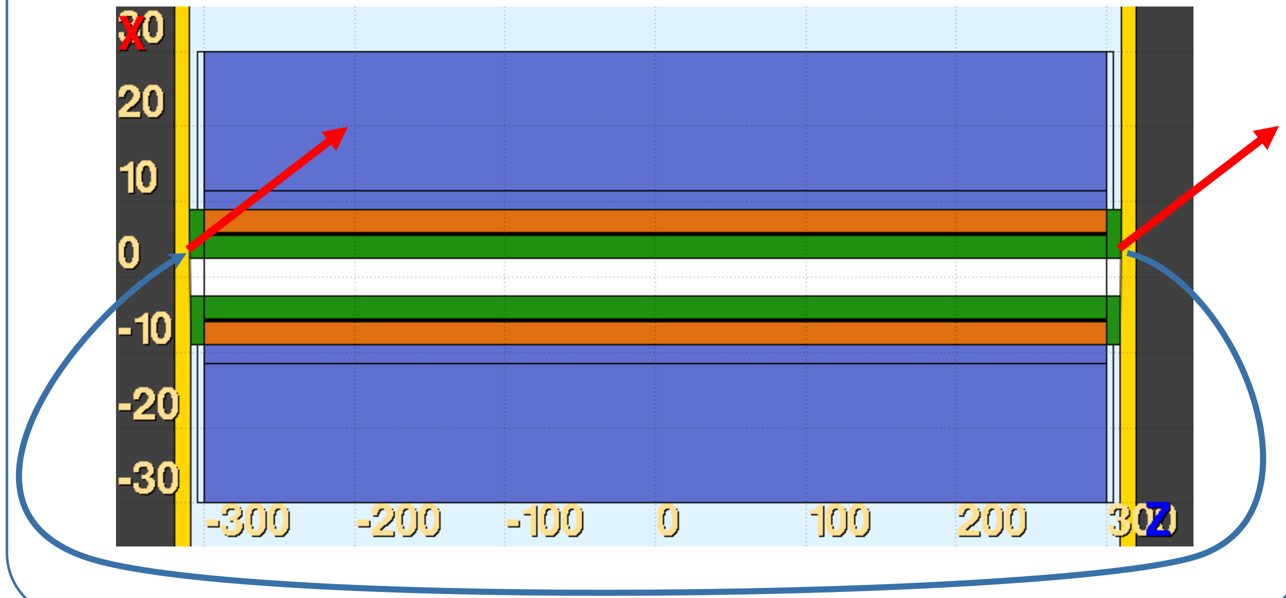
- These routines help to define the electric (**`elefld.f`**) and magnetic (**`magfld.f`**) fields in the geometry
- **`usrhsc.f`** allows the user to apply density scaling factors
- **`usrmed.f`** is a generic routine called for each material tagged via the **`MAT-PROP`** card

Non trivial magnetic fields: `magfld.f`



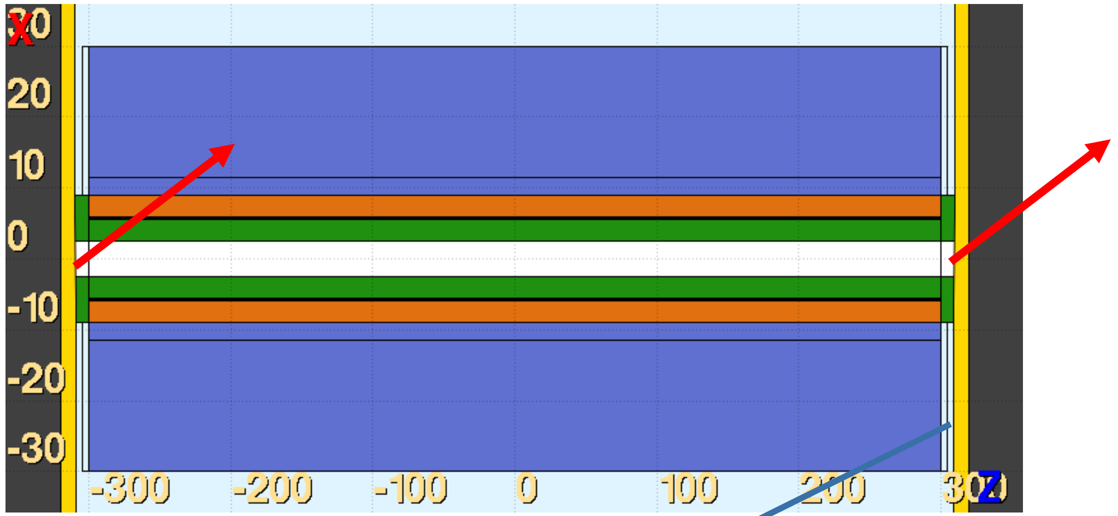
**Now also with magnetic cards!*

Particle re-injection in the geometry: `usrmed.f`



usrmed.f example (1)

- In accelerator physics, we typically have a series of elements repeated in the line; we can exploit this to minimize the geometry complexity in the input file
- Let’s take for instance an infinite series of dipoles: we can model all the dipoles in the geometry, or we can use just one element and apply a “pacman” approach, where the particles are transported on the other side when they reach the edge



Direction

Position

```
* ..+....1....+....2....+....3....+....4....+....5....+....6....+....7..
PLA epla -1.9529E-03       0.09.9999E-01        0.0        0.0309.500384
```

`usrmed.f` example (2)

- We set our region for reinjection in gold and we activate the user-defined medium routine there

```
* ..+....1....+....2....+....3....+....4....+....5....+....6....+....7..
MAT-PROP          1.0                             GOLD                    USERDIRE
```

- When a particle goes in this region, `usrmed.f` is called

```
      SUBROUTINE USRMED ( IJ, EKSCO, PLA, WEE, MREG, NEWREG, XX, YY, ZZ,
     &                    TXX, TYY, TZZ, TXXPOL, TYYPOL, TZZPOL )
* ...
*     Input variables:                                               *
*             ij = particle id                                       *
*          Eksco = particle kinetic energy (GeV)
*            Pla = particle momentum (GeV/c)
*            Wee = particle weight
*           Mreg = (original) region number
*         Newreg = (final)    region number
*       Xx,Yy,Zz = particle position
*    Txx,Tyy,Tzz = particle direction
* Txx,Tyy,Tzzpol = particle polarization direction
```

Typical commons and copyright declaration:

```
      INCLUDE 'dblprc.inc'
      INCLUDE 'dimpar.inc'
      INCLUDE 'iounit.inc'
*
*----------------------------------------------------------------------*
*                                                                      *
*     Copyright (C) 2003-2019:  CERN & INFN                            *
*     All Rights Reserved.                                             *
*                                                                      *
*     USeR MEDium dependent directives:                                *
*                                                                      *
```

usrmed.f example (3)

```
      PARAMETER ( ALPHOU = -0.0039058702941051140D+00 )
      LOGICAL LFIRST
      SAVE LFIRST, SINALP, COSALP
      DATA LFIRST / .TRUE. /
*
      IF( LFIRST ) THEN
         LFIRST = .FALSE.
         SINALP = SIN ( ALPHOU )
         COSALP = COS ( ALPHOU )
      END IF
*
      ZZNEW = -ZZ
      IF( ZZNEW .LT. ZERZER) THEN
         ZZ = ZZNEW+1D-08
      ELSE
         ZZ = ZZNEW-1D-08
      END IF
*
      DELTAX = TXX
      DELTAZ = TZZ
      TZZ    = DELTAZ * COSALP + DELTAX * SINALP
      TXX    = DELTAX * COSALP - DELTAZ * SINALP
```

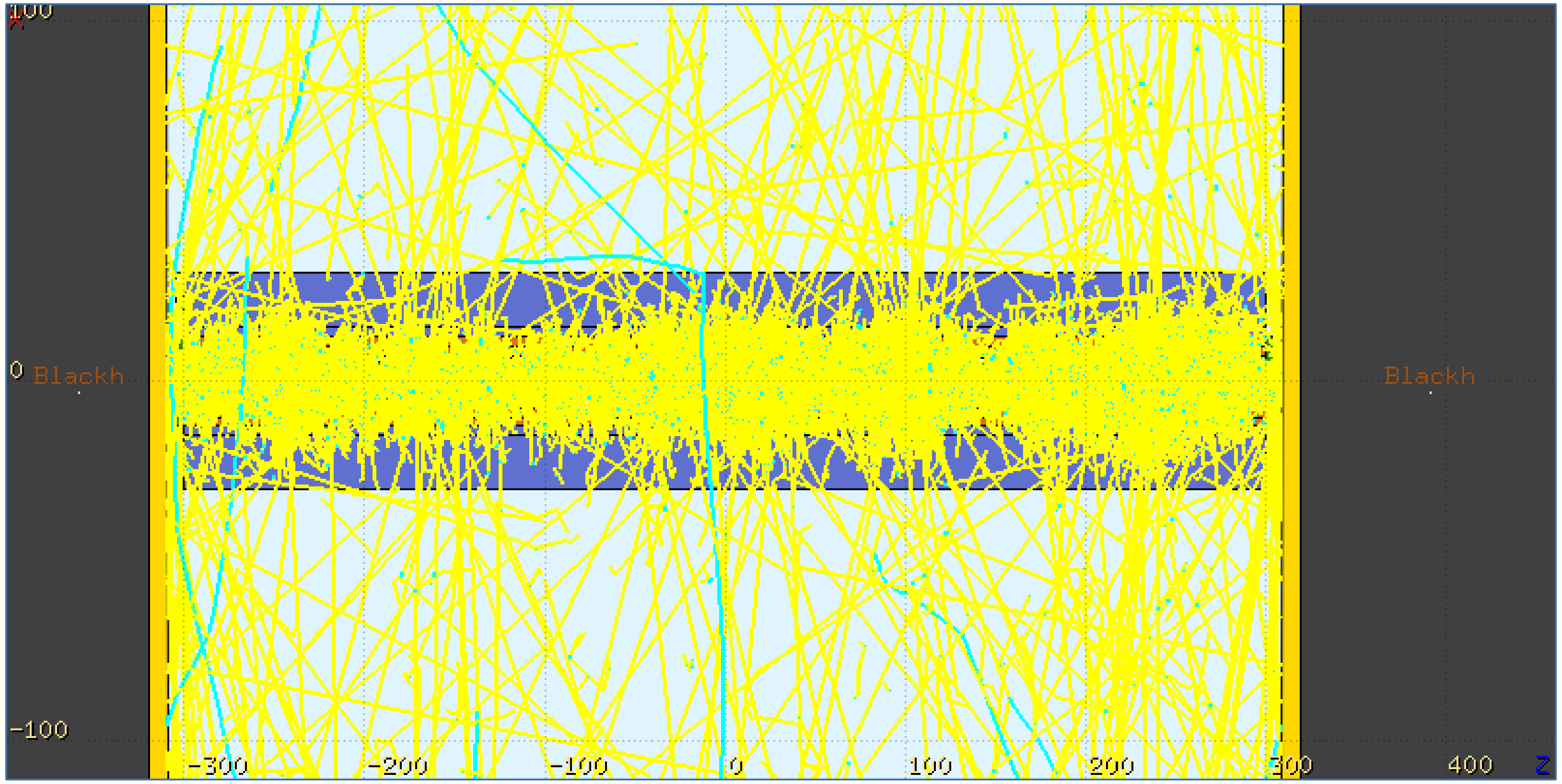
Core of the routine: angle between two consecutive dipoles

Flip z, plus a tiny tolerance to avoid numerical errors (x and y will remain the same)

Rotate the direction of the particle (and the polarization, not included here for brevity)

`usrmed.f` example (4)

- Electron and photons (down to 1 MeV) resulting from a single muon decay at 5 TeV are shown in light blue and yellow.
- The particles are re-injected many times. Huge simplification for the simulation!



User-defined stack interception

Particle stack interception

mdstck.f stupre.f
stuprf.f pshckp.f

- The first routine (**mdstck.f**) is called after a nuclear interaction, before any biasing.
- All the other routines are called before pushing FLUKA particles* (**stuprf.f**), EMF particles (**stupre.f**) or Cherenkov photons (**pshckp.f**) to the stack.

```
      SUBROUTINE MDSTCK ( IFLAG, NPSECN )
* ...
*         Iflag = 1: standard Kaskad call                      *
*               = 2: Kaskad call after elastic                 *
*               = 3: Kasneu call                               *
*               = 4: Emfsco call                               *
*-------------------------------------------------------------*
*
      INCLUDE 'emfstk.inc'
      INCLUDE 'fheavy.inc'
      INCLUDE 'genstk.inc'
      INCLUDE 'trackr.inc'
*
      RETURN
```

* *Photonuclear and electro-nuclear secondaries are always managed by* ***stuprf.f****. Synchrotron radiation photons are pushed directly to the stack.*

mdstck.f

Knowing how many particles are produced (**NPSECN**), you can access those on the secondary particle stack (**genstk.inc**)

stupre.f/stuprf.f

- These two routines are complementary and they allow the user to assign a value to one or more stack user variables when the corresponding particle is loaded into one of the stacks

By default copied (in stupre.f/stuprf.f) except in case of electromagnetic interactions

Common block:	FLKSTK	EMFSTK	OPPHST	TRACKR
LOGICAL	LOUSE	LOUEMF	LOUOPP	LLOUSE
INTEGER*11	ISPARK	IESPAK	ISPORK	ISPUSR
DOUBLE PRECISION*11	SPAREK	ESPARK	SPAROK	SPAUSR

TRACKR is accessible in mgdraw.f

Pushed to trackr.inc when the particle is transported (user does not see this part)

stupre.f/stuprf.f example: particle ancestors

- We are scoring some specific process (e.g. a background to an experiment). But where are those particles coming from?

In mgdraw.f, we save the particle species in a dedicated user variable after each particle interaction

```
      ENTRY USDRAW ( ICODE, MREG, XSCO, YSCO, ZSCO )
           ISPUSR(1) = JTRACK
      RETURN
```

In stupfr.f, we want to use the rest of the ISPUSR array to store the particle ancestors information

```
      DO 200 ISPR = 1, MKBMX2
         ISPARK (ISPR,NPFLKA) = ISPUSR (ISPR)
  200 CONTINUE
```

Default lines: they just copy `ISPUSR`

```
      DO 200 ISPR = 1, MKBMX2 - 1
         ISPARK (ISPR + 1,NPFLKA) = ISPUSR (ISPR)
  200 CONTINUE
```

The mother particle species is at `ISPR=2`
The n-th grandmother is at `ISPR=2+N`

User routines for scoring

FLUKA output

```
comscw.f fluscw.f
endscp.f fldscp.f
musrbr.f lusrbl.f
fusrbv.f usrrnc.f
```

- All these routines are used when advanced scoring is needed.
- Among the various, we will focus only on **`comscw.f`**, **`fluscw.f`** and **`usrrnc.f`**

They are activated by the card:
`USERWEIGH`

`WHAT(6) > 0`

`comscw.f`
Weighting deposited energy, stars or residual nuclei

`WHAT(3) > 0`

`fluscw.f`
Weighting fluence, current and yield

`WHAT(5) > 0`

`usrrnc.f`
Called each time a residual nucleus is produced

fluscw.f/comscw.f: structure

```
      DOUBLE PRECISION FUNCTION FLUSCW ( IJ    , PLA   , TXX   , TYY   ,
     &                                   TZZ   , WEE   , XX    , YY    ,
     &                                   ZZ    , NREG  , IOLREG, LLO   ,
     &                                   NSURF )
* ...
*     Input variables:                                                 *
*                                                                      *
*           Ij = (generalized) particle code (Paprop numbering)        *
*          Pla = particle laboratory momentum (GeV/c) (if > 0),        *
*                or kinetic energy (GeV) (if <0 )                      *
*    Txx,yy,zz = particle direction cosines                            *
*          Wee = particle weight                                       *
*     Xx,Yy,Zz = position                                              *
*         Nreg = (new) region number                                   *
*       Iolreg = (old) region number                                   *
*          Llo = particle generation                                   *
*        Nsurf = transport flag (ignore!)                              *
*                                                                      *
*     Output variables:                                                *
*                                                                      *
*       Fluscw = factor the scored amount will be multiplied by        *
*       Lsczer = logical flag, if true no amount will be scored        *
*                regardless of Fluscw                                  *
```

- Useful variables (common SCHOLP):

```
ISCRNG = 1 --> Boundary crossing estimator
ISCRNG = 2 --> Track length binning
ISCRNG = 3 --> Track length estimator
ISCRNG = 4 --> Collision density estimator
ISCRNG = 5 --> Yield estimator
JSCRNG = # of the binning/estimator
```

- This function is called just before a quantity is scored. It provides access to information about the particle which is being scored and the type of scoring.
- The user can modify **FLUSCW** (or **COMSCW**) to apply a weight different than one.

$$\begin{tikzcd} \texttt{fluscw.f} \arrow[r, leftrightarrow, "\text{specular}"] & \texttt{comscw.f} \end{tikzcd}$$

fluence-like quantitites

star-like quantitites

fluscw.f/comscw.f: example

- Another example taken from the muon collider: we have a target on which protons impact, and we need to extract a fraction of the power from it.
- How should an extraction channel look like? We want to score the particle fluence weighting it by the particle kinetic energy.

```
      DOUBLE PRECISION FUNCTION FLUSCW ( IJ     , PLA    , TXX    , TYY    ,
     &                                   TZZ    , WEE    , XX     , YY     ,
     &                                   ZZ     , NREG   , IOLREG, LLO    ,
     &                                   NSURF )
* ...
      INCLUDE 'paprop.inc'
*
      IF ( PLA .LE. ZERZER .AND. IJ .GT. 0) THEN
         E_KIN = -PLA
      ELSE IF ( IJ .GT. 0 ) THEN
         E_KIN  = SQRT ( PLA**2 + AM (IJ)**2 )
     &                   - AM (IJ)
      ELSE
         E_KIN = ZERZER
      ENDIF
      FLUSCW = ONEONE * E_KIN
      LSCZER = .FALSE.
      RETURN
```

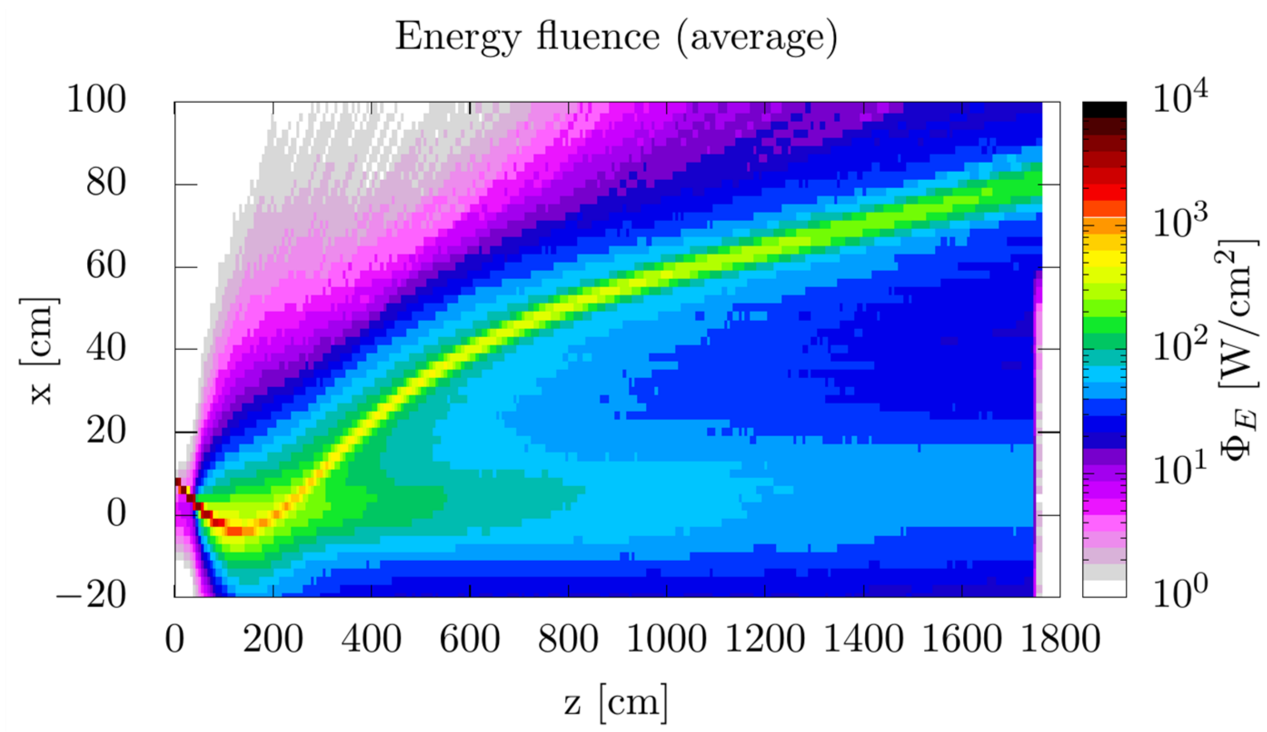
The function description is skipped here

Apply a weighting factor equal to the kinetic energy of the particle (and kill heavy ones)

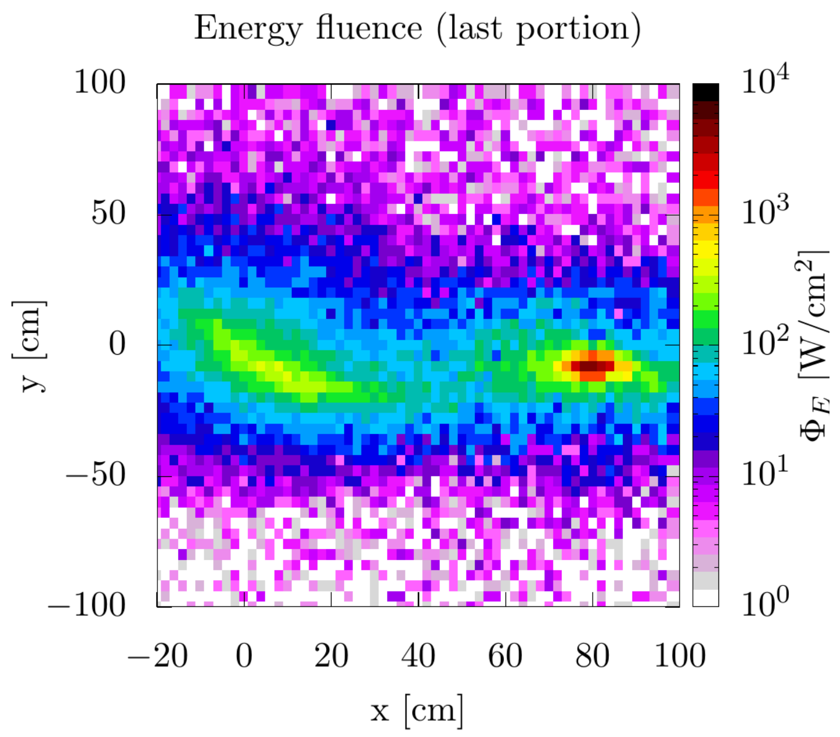
fluscw.f/comscw.f: example

```
* ..+....1....+....2....+....3....+....4....+....5....+....6....+....7..
USERWEIG                                   1
USRBIN                 10   ALL-PART     -24        100        100       1760ene_flu
USRBIN                -20       -100       0         60         50        300 &
```

Projection of the energy fluence on the xz plane.



Energy fluence at z = 1700 cm. The hotspot at x = 80 cm is where the extraction channel for the spent beam should be placed



usrrnc.f: structure

```
          SUBROUTINE USRRNC ( IZ, IA, IS, X, Y, Z, MREG, WEE, ICALL )

          INCLUDE 'dblprc.inc'
          INCLUDE 'dimpar.inc'
          INCLUDE 'iounit.inc'
* ...
*         Argument list:
*           IZ             : atomic number of the residual nucleus
*           IA             : mass number of the residual nucleus
*           IS             : isomeric state of the residual nucleus
*           X, Y, Z        : particle position
*           MREG           : number of the current region
*           WEE            : particle weight
*           ICALL          : internal code calling flag (not for general use)
```

- Subroutine **USRRNC** is called every time a residual nucleus is stopped, if option **USERWEIG** has been requested with `WHAT(5) > 0`.
- It provides all the information of the nucleus (atomic and mass number, isomeric state and position). The weight `WEE` of the residual can be used to kill it or to perform biasing
- **Warning**: biasing via weight is dangerous, since the normalisation of the results will not be managed by FLUKA

`usrrnc.f`: example

- This routine is called each time a residual nucleus is produced
- A trivial usage could be to print out the information of those residual nuclei
- Another interesting opportunity is to filter out some non-interesting radionuclides

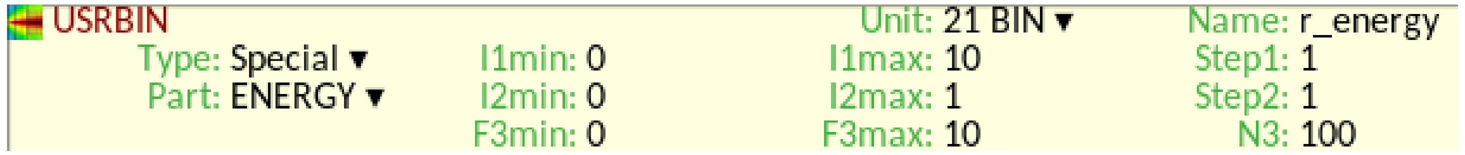
```
      SUBROUTINE USRRNC ( IZ, IA, IS, X, Y, Z, MREG, WEE, ICALL )
* ...
      IF (IZ .EQ. 27 .AND. IA .EQ. 60) THEN
         WEE = ZERZER
      ENDIF
      RETURN
```

Hardcoded variables. Can we make it more elegant with USRGCALL?

- In this very simple example, the ^{60}Co is filtered out and killed as soon as the residual nucleus is deposited. You can do also the opposite, and filter out all but one interesting nuclide

`fusrbv.f`: example radial scoring

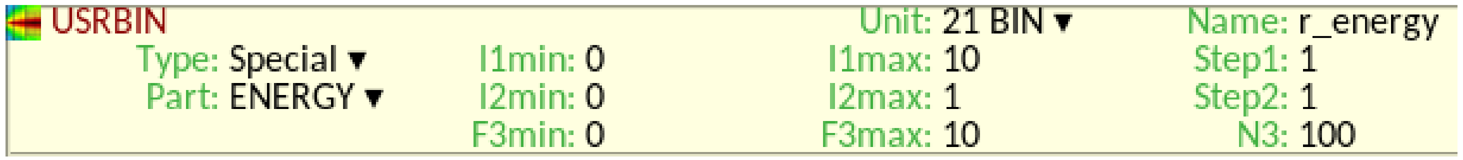
- When the user asks for a user-defined USRBIN, three routines are called: `musrbr.f`, `lusrbl.f` and `fusrbv.f`. These select the bin where the quantity is saved
- How can I ask for a radial binning?



USRBIN		Unit: 21 BIN ▾	Name: r_energy
Type: Special ▾	I1min: 0	I1max: 10	Step1: 1
Part: ENERGY ▾	I2min: 0	I2max: 1	Step2: 1
	F3min: 0	F3max: 10	N3: 100

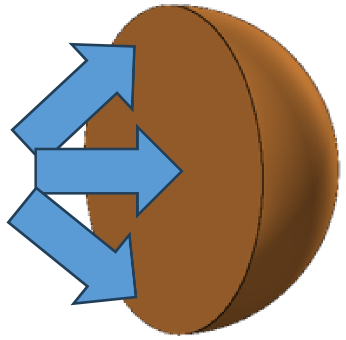
- You can ask for a user-defined USRBIN, which scores all quantities on a binning which is defined by the user.
- As of today, the 3D binning consists of two discontinuous variables (by default the region and the lattice number) and a discontinuous one (by default 0).

`fusrbv.f`: example radial scoring



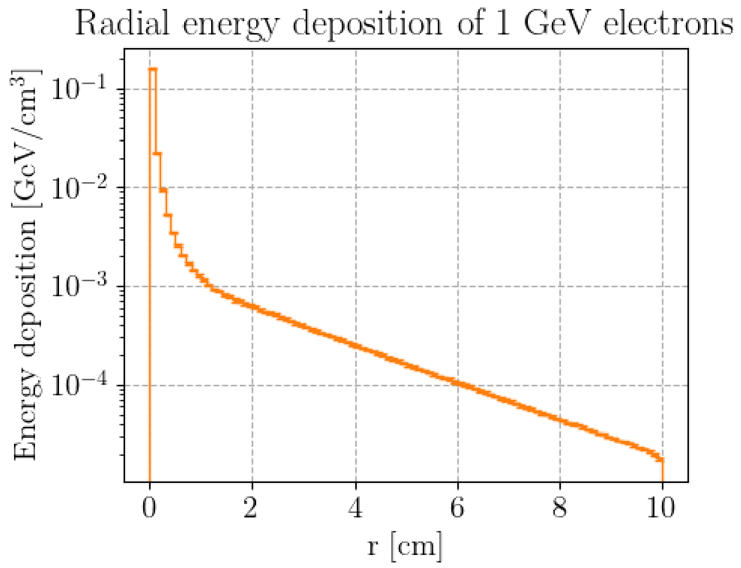
```
USRBIN                                    Unit: 21 BIN ▾          Name: r_energy
     Type: Special ▾      I1min: 0        I1max: 10               Step1: 1
     Part: ENERGY ▾       I2min: 0        I2max: 1                Step2: 1
                          F3min: 0        F3max: 10                  N3: 100
```

1 GeV electrons on a 10 cm radius half sphere



- Example: we have a 1 GeV isotropic electron beam impinging on half a sphere. We want the radial energy deposition
- To ask for a radial binning, we do not touch the first two variables (per region per lattice binning), while we modify `fusrbv.f`
- Downside: for the data analysis you are on your own!

```
      DOUBLE PRECISION FUNCTION FUSRBV ( IJ, PCONTR, XA, YA, ZA,
     &                                   MREG, ICALL )
      ...
      FUSRBV = SQRT( XA ** 2 + YA ** 2 + ZA ** 2 )
      RETURN
```



Important functions and routines

- Do not reinvent the wheel!* FLUKA offers an abundant selection of functions and routines for your usage.

* Except for didactic purposes

- **CALL FLABRT ('calling routine name','my message')**
  to abort FLUKA. To be used when an user routine reaches an unacceptable state (or for debugging!)

- **CALL OAUXFI ('file name', LUNRDB, 'OLD', IERR)**
  to open an auxiliary file (sitting in some default locations) for reading its content

- Random number generators:

`... = FLRNDM (XDUMMY)`	uniformly distributed in [0-1)
`CALL FLNRRN (RGAUSS)`	Gaussian distributed (μ=0, σ=1)
`CALL FLNRR2 (RGAUS1,RGAUS2)`	Gaussian distributed uncorrelated pair
`CALL SFECFE (SINT,COST)`	sine and cosine of uniformly distributed azimuthal angle
`CALL RACO (TXX, TYY, TZZ)`	isotropically distributed 3D direction
`CALL SFLOOD ( XXX, YYY, ZZZ, UXXX, VYYY, WZZZ )`	position and direction on a unit sphere to generate uniform and isotropic fluence **inside**

Get the region number from region name

```
CALL GEON2R ( REGNAM, NREG, IERR )
* Input variable:
* Regnam = region name (CHAR*8)
*
* Output variables:
* Nreg = region number
* Ierr = error code
* (0 on success, 1 on failure)
```

(… and vice-versa, do yo

```
CALL GEOR2N ( NREG, I
* Input variable:
* Nreg = region numbe
*
* Output variables:
* Regnam = region name (CHAR*8)
* Ierr = error code
* (0 on success, 1 on failure)
```

All **regions** are internally **treated as numbers**, both in FLUKA and user routines.

*When coding these, you should `CALL GEON2R` to *translate your region name into the respective number and save the latter for runtime use*. This has to be done only once the first time your routine is called (use `IF (LFIRST) THEN`).

Name based declaration in the inputfile

```
*Black hole
BLKBODY      5 +blkbody -void
*Void around
VOID         5 +void -target
*Target
TARGET       5 +target
```

Region numbers and names echoed in .out

```
 1  BLKBODY    1  BLCKHOLE  OFF                0.00000E+00    9.99852E+04
               (  1  BLCKHOLE OFF )
 2  VOID       2  VACUUM     OFF               0.00000E+00    9.99852E+04
               (  2  VACUUM   OFF )
 3  TARGET     12  COPPER    OFF               0.00000E+00    9.99852E+04
               ( 12  COPPER   OFF )
```

From the FLUKA environment lecture

FLUKA

Most important commons

- Always to be added:

dblprc.inc	contains as parameters most commons physical and mathematical constants. Here lies the implicit declaration for variables: IMPLICIT DOUBLE PRECISION (A-H,O-Z)
dimpar.inc	dimensions of the most important arrays
iounit.inc	logical input and output unit numbers (1 to 19 are reserved)

- Few tips:
  - Use DBLPRC parameters when possible
  - Pay attention to typos in numerical constants! With implicit declaration (as of today): TWOTHI = ⅔, TWOTHR = 0
  - If you are using a common block, do not redefine an existing variable within your routines

Other important commons

The most important commons can also be found in the FLUKA manual (13.1).
A few selected ones are:

• **BEAMCM**	properties of primary particles as defined by BEAM and BEAMPOS
• **EMFSTK**	electromagnetic stack (for e+/- and photons)
• **SOURCM**	user variables and information for a user-written source
• **FHEAVY**	stack of heavy secondaries created in nuclear evaporation
• **FLKMAT**	material properties
• **RESNUC**	properties of the current residual nucleus
• **FLKSTK**	main FLUKA particle stack
• **TRACKR**	TRACKs Recording (properties of the currently transported particle and its path)
• **PAPROP**	particle properties (masses, charges, etc.)