



# The FLUKA User Routines

---

How to tailor FLUKA  
to specific, non standard, user's needs:

User programming in the FLUKA environment



# Why User Routines

---

- Unlike some other Monte Carlo particle transport codes, Fluka gets its input mainly from a simple file. It offers a rich choice of options for scoring most quantities of possible interest and for applying different variance reduction techniques, without requiring the users to write a single line of code.
- However, although normally there is no need for any “user code”, there are special cases where this is unavoidable, either because of the complexity of the problem, or because the desired information is too unusual or too problem-specific to be offered as a standard option.
- And on the other hand, even when this is not strictly necessary, experienced programmers may like to create customised input/output interfaces.
- A number of user routines (available on LINUX and UNIX platforms in directory usermvax) allow to define non-standard input and output, and in some cases even to modify to a limited extent the normal particle transport.
- Most of them are already present in the Fluka library as dummy or template routines, and require a special command in the standard input file to be activated.
- Users can modify any one of these routines, and even insert into them further calls to their own private ones, or to external packages (at their own risk!).
- This increased flexibility must be balanced against the advantage of using as far as possible the Fluka standard facilities, which are known to be reliable and well tested.



## What is available for the users

---

- The “skeleton” of all possible user routines in `$FLUPRO/usermvax`
- The include files containing the COMMON blocks with all relevant variables to access particle stack, secondary products and their kinematics, particle and material properties, etc. in `$FLUPRO/flukapro`
- The compiling and linking scripts which are in `$FLUPRO/flutil`

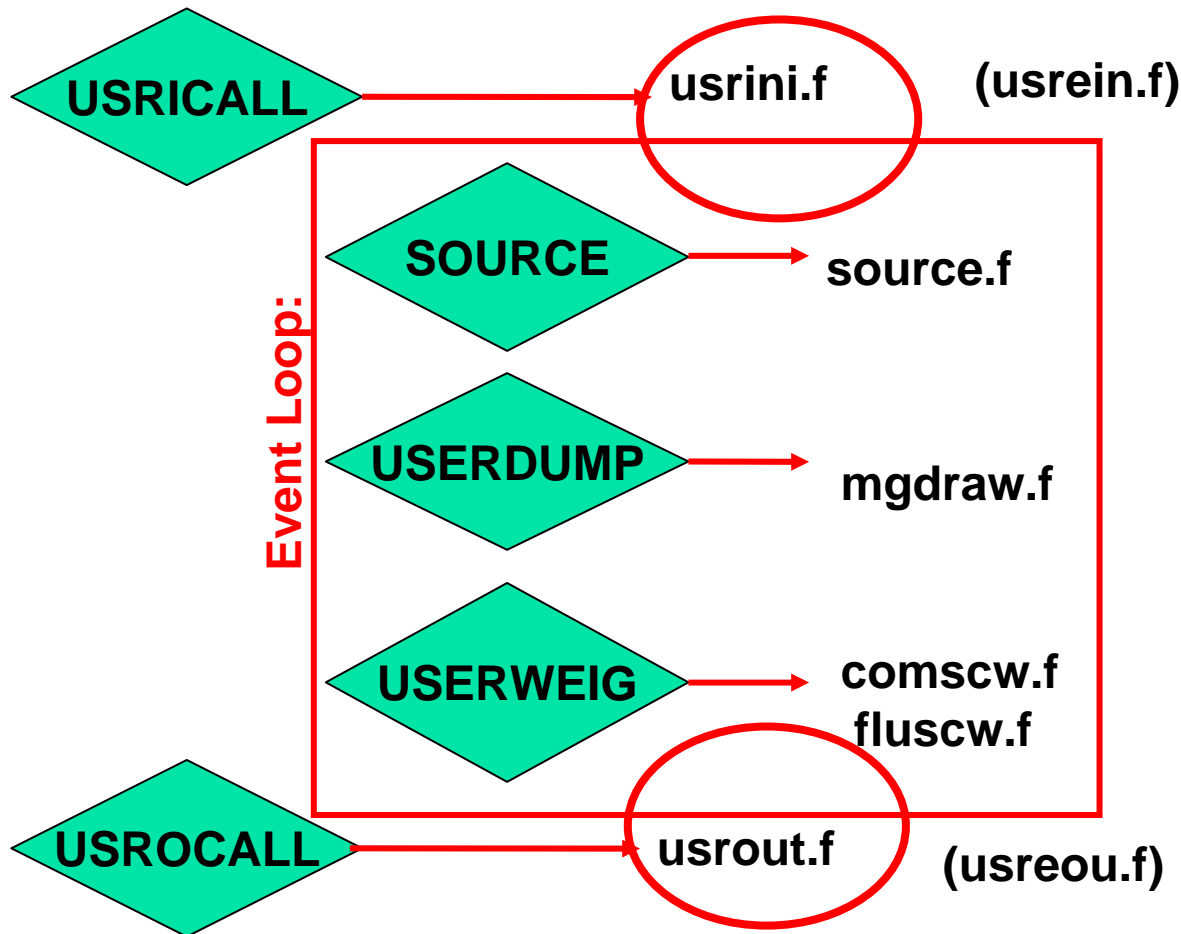


## Something more that users may need

---

- The “rules” to achieve a good programming style inside **FLUKA**
- The knowledge of a certain number of utilities already existing in **FLUKA** (mathematics, random number, general utilities, etc.)
- The “theoretical” background and the proper know-how to implement a correct algorithm for the specific user problem

# A first look to the correspondance bewteen some of the user routines and FLUKA commands



But many other correspondences exist as explained later



# The full list of user routines

---

- abscff.f
- comscw.f
- dffcff.f
- endscp.f
- fldscp.f
- fluscw.f
- formfu.f
- frghns.f
- fusrbv.f
- lattic.f
- lusrbl.f
- magfld.f
- mdstck.f
- mgdraw.f
- musrbr.f
- ophbdx.f
- pshckp.f
- queffc.f
- rflctv.f
- rfrndx.f
- soevsv.f
- source.f
- stupre.f
- stuprf.f
- absset.f
- udcdrf.f
- usimbs.f
- usrein.f
- usreou.f
- usrini.f
- usrmed.f
- usrout.f
- usrrnc.f
- ustckv.f



## A possible classification in terms of their use (1)

---

### User run control

- usrini.f
- usreini.f
- usrout.f
- usreou.f

### Event generation, physics, kinematics

- source.f
- soevsv.f
- udcdrl.f
- formfu.f

### Properties of medium

- magfld.f
- usrmed.f



See MAT-PROP



## A possible classification in terms of their use (2)

---

### in association to FLUKA output

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

### Intercepting particle stack

- mdstck.f
- stupre.f
- stuprf.f

### Biasing

- usbset.f
- usimbs.f





## A possible classification in terms of their use (3)

---

### To drive optical photon transport

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

### To manage lattice geometry

- lattic.f

### The most general output management

- mgdraw.f



See Chap. 13 of manual



# Programming in FLUKA

---

It is recommended that at least the following lines be present at the beginning of each routine:

```
INCLUDE '(DBLPRC)'  
INCLUDE '(DIMPAR)'  
INCLUDE '(IOUNIT)'
```

Each **INCLUDE** file contains a **COMMON** block, plus related constants.

Additional **INCLUDE**s may be useful, in particular **BEAMCM**, **CASLIM**, **EMFSTK**, **SOURCM**, **EVTFLG**, **FHEAVY**, **GENSTK**, **LTCLCM**, **FLKMAT**, **RESNUC**, **SCOHLP**, **SOUEVT**, **FLKSTK**, **SUMCOU**, **TRACKR**, **USRBIN**, **USRBDX**, **USRTRC**, **USRYLD**.

Note the parentheses which are an integral part of the Fluka **INCLUDE** file names.



# Programming in FLUKA

---

Directory `$FLUPRO/flukapro` contains a full documentation about the meaning of the variables of the `INCLUDE` files.

**DBLPRC:** included in all routines of Fluka, contains the declaration

**IMPLICIT DOUBLE PRECISION (A-H,O-Z)**

**DIMPAR:** dimensions of the most important arrays

**IUNIT:** logical input and output unit numbers and sets many mathematical and physical constants.

Users are strongly encouraged to adhere to “Fluka style” by using systematically double precision (except for very good reasons such as calling external single precision scoring packages), and to use constants defined in this file for maximum accuracy.



## Other important COMMON blocks in short (1)

---

- BEAMCM:** properties of primary particles as defined by options BEAM and BEAMPOS
- CASLIM:** number of primary particles followed (needed for normalisation)
- EMFSTK:** particle stack for electrons and photons
- SOURCM:** user variables and information for a user-written source
- EVTFLG:** event flags
- FHEAVY:** stack of heavy secondaries created in nuclear evaporation
- GENSTK:** properties of each secondary created in a hadronic event
- LTCLCM:** LaTtice CeLl CoMmon (needed when writing symmetry transformations for Lattice Geometry)
- FLKMAT:** material properties
- FLKSTK:** main Fluka particle stack
- RESNUC:** properties of the current residual nucleus



## Other important COMMON blocks in short (2)

---

**SCOHLP:** SCORing HeLP (information on current estimator type). It contains a flag **ISCRNG**, indicating the quantity being scored by the current estimator, and one **JSCRNG** corresponding to the binning/ detector number. Binnings and detectors are sequentially numbered according to their order of appearance in standard input. Note that several detectors can have the same **JSCRNG** number (for instance Binning N. 3 and Track-length estimator N. 3). They can be distinguished based on the value of **ISCRNG**. However, note that the same value of *ISCRNG* may have different meanings in functions *FLUSCW* and *COMSCW* (see later)



## Other important COMMON blocks in short (2)

---

**SOUEVT:** **SOU**rce **EVenT** (useful when source particles are obtained from an external event generator)

**SUMCOU:** total numbers and total weights relative to many physical and Monte Carlo events (needed for normalisation, energy balance etc.)

**TRACKR:** TRACK Recording (properties of the currently transported particle and its path)

**USRBIN, USRBDX, USRSNC, USRTRC, USRYLD:**  
parameters of the requested estimators



## Exercise and homework

---

- Look at **(DBLPRC)** and learn the available **PARAMETERS** (numerical constants, physics constants, unit conversions...)
- Look at **(IOUNIT)** and learn the default logical unit assignement in FLUKA



# usrini (USeR INItialization)

## Argument list

WHAT(1) , (2) , (3) , (4) , (5) , (6) : user-provided numerical parameters  
SDUM : user-provided character string (8 characters)

Subroutine **USRINI** is called every time a **USRICALL** card is read in input. It can be used to do any kind of initialisation: reading and manipulating data from one or more files, calling other private routines, etc.

The calling parameters can carry any kind of useful information or can be used as flags to choose between different possible actions to be performed before any particle transport takes place.





## usrein (USeR Event INitialization)

---

Subroutine **USREIN** is called just before the first source particle of an event is unloaded from stack and begins to be transported. An event is the full history of a group of related particles and their descendants.

If primaries are loaded into stack by the input option **BEAM**, there is only one source particle per event; but there can be more if the user routine **SOURCE** is used to load particles into stack. **USREIN** does not need any special command to be activated, but the default version of **USREIN** does nothing: the user can write here any kind of initialisation.



# usrou (USeR OUtput)

---

## Argument list

WHAT(1), (2), (3), (4), (5), (6) : user-given numerical parameters  
SDUM : user-given character string (8 characters)

Subroutine **USROUT** is called every time a **USROCALL** card is read in input. It is used to print special user-written output in addition to the standard one provided by default. The calling parameters can carry any kind of useful information or can be used as flags to choose between different possible actions to be performed after all particle transport has taken place.



## usreou (USeR Event OUtput)

---

Subroutine **USREOU** is called at the end of each event, namely after all event primary particles and their descendants have been transported. (*See **USREIN** above for a definition of an event*).

**USREOU** does not need any special command to be activated, but the default version of **USREOU** does nothing: the user can write here any kind of event analysis, output, etc.



# comscw (weighting energy or star deposit)

## Argument list (all variables are input only)

IJ : particle type (1 = proton, 8 = neutron, etc.: see code in 5.1)  
XA, YA, ZA : current particle position  
MREG : current geometry region  
RULL : amount to be deposited (unweighted)  
LLO : particle generation  
ICALL : internal code calling flag (not for general use)

This function is activated by option **USERWEIG** with **WHAT(6) > 0.0**. Energy and star densities obtained via **SCORE** and **USRBIN**, energy and stars obtained via **EVENTBIN** and production of residual nuclei obtained via **RESNUCLEI** are multiplied by the value returned by this function. The user can implement any desired logic to differentiate the returned value according to any information contained in the argument list (particle type, position, region, amount deposited, particle generation), or information available in **COMMON SCOHLP** (binning number, type of scored quantity). The scored quantity is given by the flag **ISCRNG** (in **SCOHLP**):

**ISCRNG = 1** –! Energy density binning

**ISCRNG = 2** –! Star density binning

**ISCRNG = 3** –! Residual nuclei scoring



## comscw (continues...)

---

The binning/detector number is given by **JSCRNG** (in **SCOHLP**) and is printed in output between the estimator type and the detector name:

Note that an detector of residual nuclei can have the same **JSCRNG** number as a binning (use the value of **ISCRNG** to discriminate).

Further information can be obtained including **COMMON TRACKR** (for instance particle's total energy, direction cosines, age). **TRACKR** contains also special user variables (both integer and in double precision) which can be used to save information about particles which have undergone some particular event.

If data concerning the current material are needed, it can be accessed as **MEDIUM(MREG)** if file (**FLKMAT**) is included.



# typical example of the use of comscw

---

A common simple application of **COMSCW** is to score dose according to the local density:

```
.....
INCLUDE '(FLKMAT)'
INCLUDE '(SCOHLP)'
.....
* ===== In order to compute doses ===== *
*
* Medium(n) is the material number of region n
* Rho(m) is the density of material m (in g/cm3)
* Iscrng = 1 means we are depositing energy (not stars)
IF ( ISCRNG .EQ. 1 ) THEN
*
*   to get dose in Gy (elcmks is the electron charge in C)
*   COMSCW = ELCMKS * 1.D12 / RHO (MEDIUM(MREG))
ELSE
*
*   oneone is defined as 1.D0 in include DBLPRC
*   COMSCW = ONEONE
ENDIF
.....
```



# Let's give a look to the (TRACKR) include

---

```
PARAMETER ( MXTRCK = 2500 )
LOGICAL LFSSSC, LPKILL
COMMON / TRACKR / XTRACK ( 0:MXTRCK ), YTRACK ( 0:MXTRCK ),
& ZTRACK ( 0:MXTRCK ), TTRACK ( MXTRCK ),
& DTRACK ( MXTRCK ), DPTRCK ( 3,MXTRCK ),
& ETRACK, PTRACK, CXTRCK, CYTRCK, CZTRCK, WTRACK,
& CXTRPL, CYTRPL, CZTRPL, ZFFTRK, ZFRTTK, ATRACK,
& CTRACK, CMTRCK, AKSHRT, AKLONG, WSCRNG, WNINOU,
& SPAUSR(MKBMX1), STTRCK, SATRCK, NTRACK, MTRACK,
& JTRACK, KTRACK, MMRCK, LT1TRK, LT2TRK, IHSPNT,
& LTRACK, LLOUSE, ISPUSR(MKBMX2), LFSSSC, LPKILL
EQUIVALENCE ( SPAUSE, SPAUSR (1) )
EQUIVALENCE ( ISPUSE, ISPUSR (1) )
SAVE / TRACKR /
```

# TRACKR (continues...)

```
*
*
*      Ntrack = number of track segments
*      Mtrack = number of energy deposition events along the track
*
* 0 < i < Ntrack
*      Xtrack = end x-point of the ith track segment
*      Ytrack = end y-point of the ith track segment
*      Ztrack = end z-point of the ith track segment
*
* 1 < i < Ntrack
*      Ttrack = length of the ith track segment
*
* 1 < j < Mtrack
*      Dtrack = energy deposition of the jth deposition event
*      Dptrck = momentum loss of the jth deposition event
*
*
*      Jtrack = identity number of the particle
*      Etrack = total energy of the particle
*      Ptrack = momentum of the particle (not always defined, if
*              < 0 must be obtained from Etrack)
*
* Cx,y,ztrck = direction cosines of the current particle
* Cx,y,ztrpl = polarization cosines of the current particle
*      Wtrack = weight of the particle
*      Wscrng = scoring weight: it can differ from Wtrack if some
*              biasing techniques are used (for example inelastic
*              interaction length biasing)
*
*      Ctrack = total curved path
*      Cmrck = cumulative curved path since particle birth
*      Zfftrk = <Z_eff> of the particle
*      Zfrttk = actual Z_eff of the particle
*      Atrack = age of the particle
*
```





# TRACKR (continues...)

```
-----  
*      Akshrt = Kshrt amplitude for K0/K0bar      *  
*      Aklong = Klong amplitude for K0/K0bar      *  
*      Wninou = neutron algebraic balance of interactions (both *  
*              for "high" energy particles and "low" energy *  
*              neutrons)                               *  
*      Spausr = user defined spare variables for the current *  
*              particle                                   *  
*      Strack = macroscopic total cross section for low energy *  
*              neutron collisions                       *  
*      Satrck = macroscopic absorption cross section for low energy *  
*              neutron collisions (it can be negative for Pnab>1) *  
*      Ktrack = if > 0 neutron group of the particle (neutron) *  
*  
*      Ntrack > 0, Mtrack > 0 : energy loss distributed along the *  
*              track                                     *  
*      Ntrack > 0, Mtrack = 0 : no energy loss along the track *  
*      Ntrack = 0, Mtrack = 0 : local energy deposition (the *  
*              value and the point are not re- *  
*              corded in Trackr)                       *  
*      Mmtrack = flag recording the material index for low energy *  
*              neutron collisions                       *
```



# TRACKR (continues...)

---

```
*          Lt1trk = initial lattice cell of the current track          *
*          (or lattice cell for a point energy deposition)            *
*          Lt2trk = final    lattice cell of the current track        *
*          Ihspnt = current geometry history pointer (not set if -1)  *
*          Ltrack  = flag recording the generation number              *
*          Llouse  = user defined flag for the current particle        *
*          Ispusr  = user defined spare flags for the current particle *
*          Lfsssc  = logical flag for inelastic interactions ending with*
*                   fission (used also for low energy neutrons)      *
*          Lpkill  = logical (user) flag for sudden particle death    *
*-----*
```



# fluscw (weighting fluence, current and yield)

## Argument list (all variables are input only)

IJ : particle type  
PLA : particle momentum (if  $> 0.0$ )  
or  $-PLA =$  kinetic energy (if  $< 0.0$ )  
TXX, TYY, TZZ : particle current direction cosines  
WEE : particle weight  
XX, YY, ZZ : particle position  
NRGFLK : current region (after boundary crossing)  
IOLREG : previous region (before boundary crossing). Useful only with  
boundary crossing estimators (for other estimators it has no  
meaning)  
LLO : particle generation  
NSURF : internal code calling flag (not for general use)

Similar to **COMSCW**. Function **FLUSCW** is activated by option **USERWEIG**, with **WHAT(3) > 0.0**. Yields obtained via **USRYIELD**, fluences calculated with **USRBDX**, **USRTRACK**, **USRCOLL**, **USRBIN**, and currents calculated with **USRBDX** are multiplied by the value returned by this function.



## fluscw (continues...)

---

The user can implement any desired logic to differentiate the returned value according to any information contained in the argument list (particle type, energy, direction, weight, position, region, boundary, particle generation), or information available in **COMMON SCOHLP** (binning or detector number, estimator type). The estimator type is given by the flag **ISCRNG** (in **COMMON SCOHLP**):

**ISCRNG** = 1 –! Boundary crossing estimator

**ISCRNG** = 2 –! Track-length binning

**ISCRNG** = 3 –! Track-length estimator

**ISCRNG** = 4 –! Collision density estimator

**ISCRNG** = 5 –! Yield estimator



## Exercise and homework

---

- Prepare the use of **COMSCW** to obtain a **USRBIN** output in local dose.
- Try to make use of **USRINI** and **USRROUT**
- Give a look to **(FLKMAT)** and **(SCOHLP)**



# magfld (definition of a magnetic field)

## Argument list

X, Y, Z : current position (input only)  
BTX, BTY, BTZ : direction cosines of the magnetic field vector (returned)  
B : magnetic field intensity in tesla (returned)  
NREG : current region (input only)  
IDISC : if returned = 1, the particle will be discarded

**MAGFLD** is activated by option **MGNFIELD** with **WHAT(4-6)=0.0** and is used to return intensity and direction of a magnetic field based on the current position and region. It is called only if the current region has been flagged as having a non-zero magnetic field by option **ASSIGNMAT**, with **WHAT(5) = 1.0**.

The magnetic field spatial distribution is often read and interpolated from an external field map.



# Beware of the normalization of direction cosines!

---

**Note that in any case the direction cosines must be properly normalised in double precision even if  $B = 0.0$ .**

**The recommended algorithm is:**

```
BINLEN = ONEONE/SQRT(BTX**2+BTY**2+BTZ**2)
BTX = BTX * BINLEN
BTY = BTY * BINLEN
BTZ = BTZ * BINLEN
```

**The need for an high accuracy normalization of direction cosines is a general rule in FLUKA: there are checks that in case of bad normalization produce a lot of error messages (and time loss...)**



# source (user written source: generation of initial kinematics)

## Argument list

**NOMORE** : if set = 1, no more calls will occur (the run will be terminated after exhausting the primary particles loaded onto stack in the present call). The history number limit set with option **START** will be overridden

Subroutine **SOURCE** is probably the most frequently used user routine. It is activated by option **SOURCE** and is used to sample primary particle properties from distributions (in space, energy, time, direction or mixture of particles) too complicated to be described with the **BEAM**, **BEAMPOS** and **BEAMAXES** cards alone. For each phase-space variable, a value must be loaded onto **COMMON FLKSTK** (particle bank) before returning control. These values can be read from a file, generated by some sampling algorithm, or just assigned.





# Using source

---

Option **SOURCE** allows the user to input up to 12 numerical values (**WHASOU(1),(2). . . (12)**) and one 8-character string (**SDUSOU**) which can be accessed by the subroutine by including the following line:

**INCLUDE '(SOURCM)'**

**The user can insert any initialization within the following IF block:**

```
* +-----*
* | First call initialisations:
* |   IF ( LFIRST ) THEN
* |   *** The following 3 cards are mandatory ***
* |       TKESUM = ZERZER
* |       LFIRST = .FALSE.
* |       LUSSRC = .TRUE.
* |   *** User initialisation ***
* |   END IF
* |
* +-----*
```



## Using source (continues...)

---

The user can load onto the FLKSTK stack one or more source particles at each call: for each particle loaded the pointer must be increased by 1.

```
NPFLKA = NPFLKA + 1           ! increases the pointer
```

sets the weight of the particle = 1.0

```
WTFLK (NPFLKA) = ONEONE
```

update the total weight of the primaries (don't change):

```
WEIPRI = WEIPRI + WTFLK (NPFLKA)
```

by default sets the type of particle equal to the one defined by the BEAM card. If no BEAM card is given in input, IJBEAM is = 1 (proton).

```
ILOFLK (NPFLKA) = IJBEAM
```



# Using source (continues...)

---

## Never change the following lines:

```
* From this point ....
    LOFLK (NPFLKA) = 1           ! Generation is 1 for source particles
    LOUSE (NPFLKA) = 0           ! User variables: the user can set
    DO 100 ISPR = 1, MKBMX1      ! different values in the STUPRF or
        SPAREK (1,NPFLKA) = ZERZER ! STUPRE routine, but it is better
100  CONTINUE                   ! not to do it here
    DO 200 ISPR = 1, MKBMX2
        ISPARK (ISPR,NPFLKA) = 0 ! More user variables (integer)
200  CONTINUE
    NPARMA = NPARMA + 1         ! Updating the maximum particle number
    NUNPAR (NPFLKA) = NPARMA    ! Setting the current particle number
    NEVENT (NPFLKA) = 0         ! Resetting the current event number
    DFNEAR (NPFLKA) = +ZERZER   ! Resetting the distance to the
                                ! nearest boundary
* ... to this point: don't change anything
```





## Using source (continues...)

---

### Assigning momentum/energy

In the template routine, the momentum is assumed to be assigned by **BEAM** option (its value, **PBEAM**, is taken from **COMMON BEAMCM**, which contains all values defined by options **BEAM** and **BEAMPOS**).

```
PMOFLK (NPFLKA) = PBEAM
```

```
TKEFLK (NPFLKA) = SQRT ( PBEAM**2 + AM (IJBEAM)**2 ) - AM (IJBEAM)
```

Here the user can make direct assignment of momentum, read from an external file or introduce code lines to sample from a distribution

Alternatively the user can make sample kinetic energy and derive the momentum. **Be coherent!**

```
TKEFLK (NPFLKA) = ENSAMP
```

```
PMOFLK (NPFLKA) = SQRT(ENSAMP * (ENSAMP + TWOTWO * AM(IJBEAM)))
```



## Using source (continues...)

---

### Here direction cosines are assigned/read/sampled:

```
TXFLK (NPFLKA) = UBEAM           ! Assumed here to be the same as
TYFLK (NPFLKA) = VBEAM           ! defined by option BEAMPOS. UBEAM,
TZFLK (NPFLKA) = WBEAM           ! VBEAM, WBEAM are some among the beam
                                   ! properties in COMMON BEAMCM
```

**Be careful to ensure the proper normalization within machine accuracy!! (see the case of MAGFLD)**

### It is also possible to assign a polarization:

```
TXPOL (NPFLKA) = -TWOTWO         ! -2 is a flag for "no polarisation"
TYPOL (NPFLKA) = +ZERZER
TZPOL (NPFLKA) = +ZERZER
```

### Finally, initial space coordinate assigned/read/sampled:

```
XFLK (NPFLKA) = XBEAM           ! Assumed here to be the same as
YFLK (NPFLKA) = YBEAM           ! defined by option BEAMPOS. XBEAM,
ZFLK (NPFLKA) = ZBEAM           ! YBEAM, ZBEAM are also in COMMON BEAMCM
```



## Using source (continues...)

---

The last line calls the **SOEVSU** user routine to save the stack for possible further use.

The values of beam characteristics defined by commands **BEAM** and **POLARIZAti** are available in **COMMON BEAMCM**: the angular divergence (variable **DIVBM**), beam width (**XSPOT** and **YSPOT**), and the polarisation vector (**UBMPOL**, **VBMPOL**, **WBMPOL**) can help to set up a scheme to sample the corresponding quantities from user-defined distributions. But sampling from the distributions pre-defined by **BEAM** and **POLARIZAti** is not simply inherited by subroutine **SOURCE**: it is the responsibility of the user to write such a scheme! For this task, it may be useful to define a “beam reference frame” by means of option **BEAMAXES**.



## Using source (continues...)

---

- **Very Important:**

when using SOURCE, remember that the use of the BEAM card remains mandatory:

In that case the momentum (or energy) in WHAT(1) is meant as the maximum possible value of momentum (or energy) of your problem: this is used by FLUKA at initialization time to perform tabulations.

If, at run time, an energy greater than the maximum established in BEAM is sampled, an abort will be generated.





## Exercise and homework

---

- Study the **BEAMCM**, **SOURCM**, **SOUEVT** commons
- Write your own **source** to generate a gaussian spread in energy of a beam (possible also just with **BEAM** card, but just to verify your capabilities...)



# mgdraw (general event interface)

The most general interface to FLUKA content (if you know how to use it...)

```
Argument list (all variables are input only)
ICCODE : FLUKA physical compartment originating the call
         = 1: call from subroutine KASKAD (hadrons and muons)
         = 2: call from subroutine EMFSCO (e-, e+ and photons)
         = 3: call from subroutine KASNEU (low-energy neutrons)
         = 4: call from subroutine KASHEA (heavy ions)
         = 5: call from subroutine KASOPH (optical photons)
MREG   : current region
```

Subroutine **MGDRAW**, activated by option **USERDUMP** with **WHAT(1) ≥ 100.0**, usually writes a “collision tape”, i.e., a file where all or selected transport events are recorded. The default version (unmodified by the user) offers several possibilities, selected by **WHAT(3)** in **USERDUMP**.



## mgdraw (continues...)

---

The different **ENTRY** points of **MGDRAW**

Additional flexibility is offered by a user entry **USDRAW**, interfaced with the most important physical events happening during particle transport.

The user can modify of course also any other entry of this subroutine:

**BXDRAW** called at boundary crossings,

**EEDRAW** called at event end,

**MGDRAW** for trajectory drawing,

**ENDRAW** for recording of energy deposition events

**SODRAW** for recording of source events):



## mgdraw (continues...)

---

Possibilities the format of the output file can be changed, and different combinations of events can be written to file.

But the most interesting aspect of the routine is that the six entries (all of which, if desired, can be activated at the same time by setting `USERDUMP` with `WHAT(3)=0.0` and `WHAT(4)≥1.0`) constitute a complete interface to the whole Fluka transport. Therefore, `MGDRAW` can be used not only to write a collision tape, but to do any kind of complex analysis. Typical: event by event output (common for HEP applications).





# mgdraw: the MGDRAW entry

---

- MTRACK:** number of energy deposition events along the track
- JTRACK:** type of particle
- ETRACK:** total energy of the particle
- WTRACK:** weight of the particle
- NTRACK:** values of **XTRACK**, **YTRACK**, **ZTRACK**: end of each track segment
- MTRACK:** values of **DTRACK**: energy deposited at each deposition event
- CTRACK:** total length of the curved path

Other variables are available in **TRACKR** (but not written by **MGDRAW** unless the latter is modified by the user: particle momentum, direction cosines, cosines of the polarisation vector, age, generation, etc. (see a full list in the comment in the **INCLUDE** file).



# mgdraw: the BXDRAW entry

---

Called at Boundary Crossings

Argument list (all variables are input only)

ICODE : physical compartment originating the call, as in the MGDRAW entry  
MREG : region from which the particle is exiting  
NEWREG : region the particle is entering  
XSCO, YSCO, ZSCO : point where the boundary crossing occurs



# mgdraw: the EEDRAW entry

---

Called at Event End

**Argument list** (all variables are input only)

**ICODE** : physical compartment originating the call, as in the MGDRAW entry





# mgdraw: the ENDRAW entry

Called at  
pointlike Energy  
Deposition dumps

(for example:  
stopping particles,  
photoelectric eff.,  
etc.)

## Argument list (all variables are input only)

ICODE : type of event originating energy deposition  
ICODE = 1x: call from subroutine KASKAD (hadrons and muons);  
= 10: elastic interaction recoil  
= 11: inelastic interaction recoil  
= 12: stopping particle  
= 14: particle escaping (energy deposited in blackhole)  
ICODE = 2x: call from subroutine EMFSCO (electrons, positrons and photons)  
= 20: local energy deposition (i.e. photoelectric)  
= 21 or 22: particle below threshold  
= 23: particle escaping (energy deposited in blackhole)  
ICODE = 3x: call from subroutine KASNEU (low-energy neutrons)  
= 30: target recoil  
= 31: neutron below threshold  
= 32: neutron escaping (energy deposited in blackhole)  
ICODE = 4x: call from subroutine KASHEA (heavy ions)  
= 40: ion escaping (energy deposited in blackhole)  
ICODE = 5x: call from subroutine KASOPH (optical photons)  
= 50: optical photon absorption  
= 51: optical photon escaping (energy deposited in blackhole)  
MREG : current region  
RULL : energy amount deposited  
XSCO, YSCO, ZSCO : point where energy is deposited



# mgdraw: the SODRAW entry

Argument list

No arguments

**SODRAW** writes by default, for each source or beam particle:

**-NCASE:** (in **COMMON CASLIM**, with a minus sign to identify **SODRAW** output) number of primaries followed so far

**NPFLKA:** (in **COMMON FLKSTK**) stack pointer

**NSTMAX:** (in **COMMON FLKSTK**) highest value of the stack pointer encountered so far

**TKESUM:** (in **COMMON SOURCM**) total kinetic energy of the primaries of a user written source, if applicable. Otherwise = 0.0

**WEIPRI:** (in **COMMON SUMCOU**) total weight of the primaries handled so far

NPFLKA times:  
(all variables in  
COMMON FLKSTK)

ILOFLK:	type of source particle
TKEFLK + AM:	total particle energy (kinetic+mass)
WTFLK:	source particle weight
XFLK, YFLK, ZFLK:	source particle position
TXFLK, TYFLK, TZFLK:	source particle direction cosines



# mgdraw: the USDRAW entry

USDRAW is called  
after each  
particle interaction  
(requested by the  
user with option  
USERDUMP,  
WHAT(4)≥1.0)

## Argument list (all variables are input only)

```
ICODE : type of event
ICODE = 10x: call from subroutine KASKAD (hadron and muon interactions);
      = 100: elastic interaction secondaries
      = 101: inelastic interaction secondaries
      = 102: particle decay secondaries
      = 103: delta ray generation secondaries
      = 104: pair production secondaries
      = 105: bremsstrahlung secondaries
ICODE = 20x: call from subroutine EMFSCO (electron, positron and photon interactions)
      = 208: bremsstrahlung secondaries
      = 210: Møller secondaries
      = 212: Bhabha secondaries
      = 214: in-flight annihilation secondaries
      = 215: annihilation at rest secondaries
      = 217: pair production secondaries
      = 219: Compton scattering secondaries
      = 221: photoelectric secondaries
      = 225: Rayleigh scattering secondaries
ICODE = 30x: call from subroutine KASNEU (low-energy neutron interactions)
      = 300: neutron interaction secondaries
ICODE = 40x: call from subroutine KASHEA (heavy ion interactions)
      = 400: delta ray generation secondaries

MREG  : current region
XSCO, YSCO, ZSCO : interaction point
```



## Exercise

---

- Give a look at COMMONS: FLKSTK, GENSTK, EMFSTK, FHEAVY
- Make use of entry BXDRAW of mgdraw.f to prepare an event by event output at a boundary crossing
- Look at the examples on the web site to understand how to use mgdraw.f to create outputs in other environments (for instance HBOOK, Root)



# When mgdraw should never be used

---

- When biasing is requested
- Whenever low-energy neutrons ( $E < 20$  MeV) are used

(or at least one has to be a real very experienced user to manage these cases without making mistakes...)



# Using Random Numbers in user routines

---

- **... = FLRANDM(XDUMMY)**

returns a 64-bit random number (0-1)

- **CALL FLNRRN (RGAUSS)**

returns a normally distributed random number **RGAUSS**

- **CALL FLNRR2 (RGAUS1,RGAUS2)**

returns an uncorrelated pair of normally distributed random numbers: **RGAUS1** and **RGAUS2**

- **CALL SFECFE(SINT,COST)**

returns a pair of random numbers, **SINT** and **COST** such that:  **$SINT^{**2}+COST^{**2} = 1.D+00$**

- **CALL RACO (TXX, TYY, TZZ)**

returns 3 random numbers (**TXX**, **TYY**, **TZZ**) such that:  **$TXX^{**2}+TYY^{**2} + TZZ^{**2}= 1.D+0$**



# Mathematical library in FLUKA

---

- **FLUKA contains many mathematical routines of general utility, so in general there should not be the need to call external mathematical libraries:**

<b>flgaus:</b>	<b>Gaussian adaptative integration</b>
<b>simpsn:</b>	<b>Simpson integration</b>
<b>gamfun:</b>	<b>Gamma fuction</b>
<b>radcub:</b>	<b>Real solutions of 3<sup>rd</sup> order algebraic equation</b>
<b>flgndr:</b>	<b>Legendre polinomials</b>
<b>yinter, finter:</b>	<b>interpolation routines</b>
<b>splinw:</b>	<b>Spline interpolation</b>
<b>rordin, rordde:</b>	<b>Sorting of vector values</b>

...

**Also: expansion in Laguerre and Chebishev polynomials, Bezier fit, and many others...**

**At some time it will be possible to have a short-writeup for their use.**



## Other useful routines

---

- **CALL OAUXFI ('file', LUN, 'CHOPT', IERR)**

to open an auxiliary file (to read data or parameters) looking automatically for the file in some default locations (temporary directory, working directory, **\$FLUPRO**, **\$HOME**) without the need of giving the full path. (see also **OPEN** data card)

- **CALL FLABRT('name','message')**

this allows to force a FLUKA abort on user request: it might be useful to perform a debugging (using gdb for instance)





# Compiling and linking FLUKA user routines

---

Each user routine in FLUKA loads commons in the include subdirectory `$FLUPRO/flukapro/` etc.

`$FLUPRO/flutil/fff` has the right definition of the path to the include subdirectory and the necessary list of g77 options

Example: `$FLUPRO/flutil/fff usrini.f` generates `usrini.o`


then `$FLUPRO/flutil/fluka -m fluka usrini.o -o flukamy` will perform the proper linking generating the executable here called `flukamy`

Tip: `$FLUPRO/flutil/fluka -m fluka usrini.f -o flukamy` will automatically call `$FLUPRO/flutil/fff`



# Producing a user's library

- Sometimes the number of user routines may be large. In this case the use of an object collection (library) is advisable
- Ifluka is organized so as to look for libraries called: libnnnn.a
- Typically the steps are the following:
  1. compile with **fff** each user routine producing the **\*.o** objects
  2. build the library: **ar -rv libnnnn.a \*.o**
  3. link with the command: **\$FLUPRO/flutil/Ifluka -m fluka -o name\_executable -O nnnn**



***-O has to be used when adding objects and in particular when replacing modules which already exist in libflukahp.a (as the user routines)***