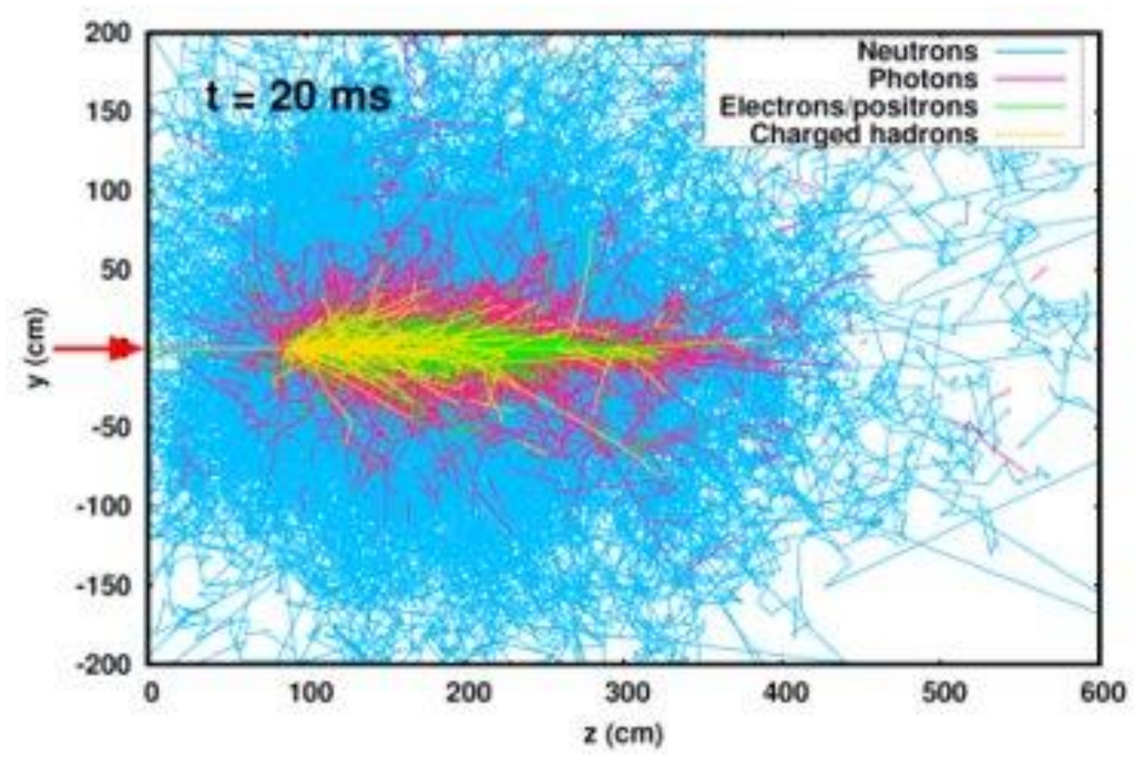
FLUKA user routines II: MGDRAW and its entries

Outline

Introduction

- The FLUKA user can be interested in quantities at a high level of granularity, while the standard FLUKA cards provide quantities collected along the entire run.



Selection of particles trajectories in a 450 GeV proton on Al simulation

- With MGDRAW, FLUKA offers the user with capability to **dump information for "events" of interest**. This lecture gives a basic introduction on the MGDRAW user routine, what it offers, how to use it.

Example user needs

Several user needs are not covered by the standard cards presented in the FLUKA Beginner course.

One might for example want to:

- Perform a **non-standard FLUKA scoring**
  While in general this is not recommended, because the available FLUKA scoring facilities are reliable, efficient and well tested, there are special cases where a user-written scoring is necessary.
- Save details of FLUKA transport, for **a new independent analysis**.
- Perform some **manipulation in an intermediate phase of Monte Carlo transport**
  The transport problem can be split into two (or more) sequential phases: (possibly post-processed) output quantities from FLUKA transport can be re-injected as source of a consecutive FLUKA run.
  Example: Splitting extension. Record all particles crossing a given boundary. Then, in a successive run, sample repeatedly source particles from that record (with a SOURCE routine).
- **Interface FLUKA to other radiation transport codes**
- **Visualize trajectories or events in a GUI**
- **FLUKA transport debugging**

What are MGDRAW and USERDUMP?

- In all cases, **the user would like to access information on “events” of interest**.
  The "event" can be a specific interaction, a boundary-crossing event, a local energy deposition event, etc.

- **MGDRAW is a FLUKA subroutine, which can provide information on the source particles, trajectories, continuous and local (point-like) energy deposition events, boundary-crossing events, as well as on user-defined events**.

- NB: In the past, the expression “collision tape” was used, but it is slightly restrictive.

- MGDRAW can be **activated by the FLUKA card: `USERDUMP`**
  When activated, MGDRAW subroutine (or its entries) is directly called from FLUKA code core.

MGDRAW: standard versus user implementations

- Like every user subroutine, MGDRAW has a **default implementation**, which is an integrated part of FLUKA source.
  It is located in: **`src/user/mgdraw.f`**
  If no user implementation is provided, it is the implementation to be called by FLUKA.

- If ever a **user implementation** is provided (**`mgdraw.f`** is modified by the user), the **user implementation is considered *instead* of the default one**.
  The user implementation is compiled and linked to create a custom executable, where the user functions definitions will overwrite the default ones.
  To help the user with a custom implementation, a skeleton is provided in:
  `src/user/mgdraw_empty.f`

- As a rule of thumb, the user should **always favour using the default FLUKA cards** (extensively debugged and cover most use cases!), or the default implementation, rather than a dedicated implementation.

Overview of MGDRAW and its entries

- `mgdraw.f` contains 6 subroutines:

→ **SODRAW**: source particles	[default implementation: dumps all]
→ **MGDRAW**: trajectories, and continuous energy losses	[default implementation: dumps all]
→ **ENDRAW**: local (point-like) energy deposition events	[default implementation: dumps all]
→ **BXDRAW**: boundary-crossing events	[default implementation: empty]
→ **USDRAW**: user-defined dumps after interactions	[default implementation: empty]
→ **EEDRAW**: end of events (1 event = 1 primary history)	[default implementation: empty]

- These entries are called from the **most important physical events happening during particle transport**, where relevant information is available and can be "observed".

The USERDUMP card

Activates FLUKA calls to MGDRAW entries, in order to dump a collision tape.

SDUM != UDQUENCH

```
*  ..+....1....+....2....+....3....+....4....+....5....+....6....+....7..
USERDUMP             100.                      2                       name
```

- **WHAT(1):** **≥ 100.0 : <u>General activation</u> of FLUKA calls to MGDRAW and/or its entries**
  < 0.0: The default is reset, i.e., no dump is written.
  = 0.0: Ignored
  > 0.0 and < 100.0: Not allowed (legacy)!

- **WHAT(2):** **Number of the unformatted output unit, when the MGDRAW default version is used**
  <u>NB:</u> When a user-implemented MGDRAW is used, the unit number can be defined by an explicit Fortran OPEN statement in MGDRAW code.
  <u>Reminder:</u> Avoid < 21.0 values (possible conflicts with Fluka pre-defined units).
  (Default = 49.0)

The USERDUMP card

Activates FLUKA calls to MGDRAW entries, in order to dump a collision tape.

SDUM != UDQUENCH

```
*  ..+....1....+....2....+....3....+....4....+....5....+....6....+....7..
USERDUMP                  100.                       2                     name
```

- **WHAT(3):** **Selection of MGDRAW entries to be called** (provided WHAT(1) ≥ 100).
  See next slide.
- **WHAT(4):** ≥ 1.0: **Activates calls to USDRAW and EEDRAW entries** (provided WHAT(1) ≥ 100)
  = 0.0: Ignored
  < 0.0: Resets to default
- **WHAT(5) – WHAT(6):** **Not used**
- **SDUM:** **Output file name** (max. 10 characters).
  NB: When a user-implemented MGDRAW is used, the user can define a longer name by an explicit Fortran OPEN statement in the MGDRAW code.

The USERDUMP card

Activates FLUKA calls to MGDRAW entries, in order to dump a collision tape.

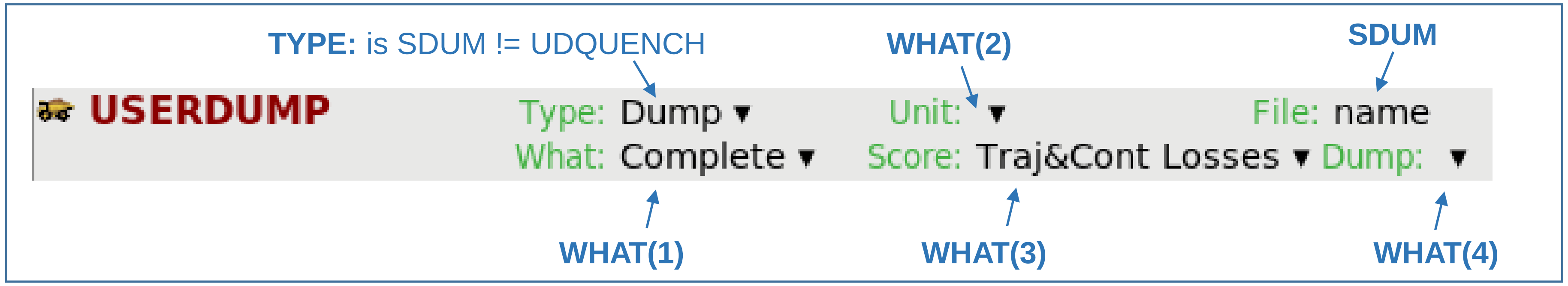
SDUM != UDQUENCH

```
*  ..+....1....+....2....+....3....+....4....+....5....+....6....+....7..
USERDUMP                  100.                           2                     name
```

- **WHAT(3):** **Selection of MGDRAW entries to be called** (provided WHAT(1) ≥ 100)
  (Default = 0)
  ≤ 0.0: Call to SODRAW every time a source particle is started, to MGDRAW at each particle step and each continuous energy loss, to **ENDRAW** at each local energy loss, to BXDRAW at each boundary crossing, and to EEDRAW at each end of event.
  → **With default MGDRAW implementation**: Source particles, trajectories, continuous and local energy losses are **ALL** dumped.
  ≥ 7.0: No call to SODRAW, MGDRAW, **ENDRAW**, **BXDRAW**, EEDRAW (calls to USDRAW and EEDRAW by WHAT(4) are unaffected).
  → **With default MGDRAW implementation**: Source particles, trajectories, continuous and local energy losses are **NOT** dumped.

The USERDUMP card

- **With Flair:**



USERDUMP Type: Dump ▾ Unit: ▾ File: name
What: Complete ▾ Score: Traj&Cont Losses ▾ Dump: ▾

- **Examples:**

Write a binary file called "name", pre-connected to the default logical output unit 49, and containing all trajectories and continuous energy losses.

```
*  ..+....1....+....2....+....3....+....4....+....5....+....6....+....7..
USERDUMP                 100.        49          2                        name
```

Write a binary file called "MYSECS", pre-connected to the logical output unit 50, and containing user-defined dumps after collisions, and ends of events.

```
*  ..+....1....+....2....+....3....+....4....+....5....+....6....+....7..
USERDUMP                 100.        50          7         1            MYSECS
```

MGDRAW entries: MGDRAW

Trajectories and continuous energy losses

default implementation

```
      SUBROUTINE MGDRAW ( ICODE, MREG )
...
      WRITE (IODRAW) NTRACK, MTRACK, JTRACK,
     &               SNGL (ETRACK), SNGL (WTRACK)
      WRITE (IODRAW) ( SNGL (XTRACK (I)), SNGL (YTRACK (I)),
     &               SNGL (ZTRACK (I)), I = 0, NTRACK ),
     &             ( SNGL (DTRACK (I)), I = 1, MTRACK ),
     &               SNGL (CTRACK)
...
      RETURN
```

input variables:

ICODE : 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

MGDRAW (when activated) writes ***by default***, for **each trajectory**, the following variables:

output variables:

NTRACK : number of track segments
MTRACK : number of continuous energy deposition events along the track. Local energy deposition events, i.e., energy deposition at a point, such as that from heavy recoils, particles below threshold and low energy neutron kerma, are written instead by the entry ENDRAW (see below)
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

trackr.inc

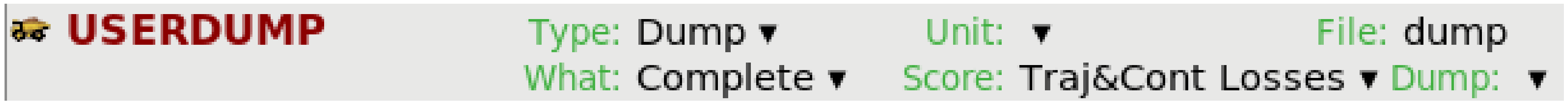
Example: plotting trajectories with Flair

See Ex2 in exercise session.
We will follow this procedure together.
Note here the use of default MGDRAW.

- **We want to visualize the particles trajectories within an event.**

- MGDRAW allows the **dump of trajectory information** in a format **compatible with Flair**.
- We are going to rely on the **default version of MGDRAW**: no need to modify any code within **mgdraw.f**.

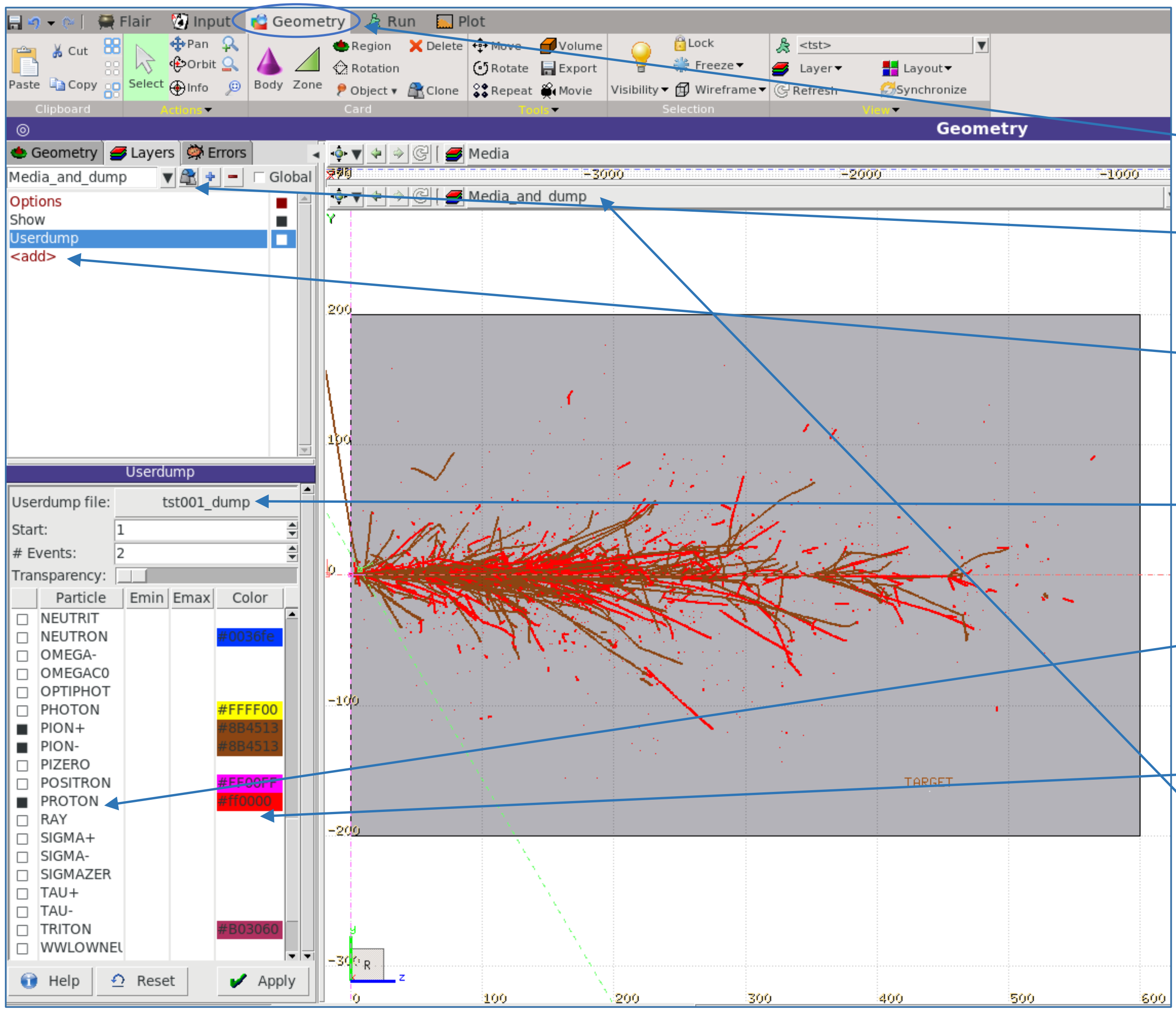
- (1) **Activate MGDRAW calls**
  To do so, add a USERDUMP card to your project with trajectories dump enabled ("Traj&Cont losses").
  It is advised to use a file name containing the word "dump" (for Flair to detect it easily in the Geometry tab).



USERDUMP Type: Dump ▾ Unit: ▾ File: dump
What: Complete ▾ Score: Traj&Cont Losses ▾ Dump: ▾

- (2) **Run your simulation (default FLUKA executable).**
  WARNING: Try with 1 primary first (output file size)!
  - NB: You cannot "Process" dump files.

Example: plotting trajectories with Flair



Userdump

Userdump file: tst001_dump
Start: 1
Events: 2
Transparency:

	Particle	Emin	Emax	Color
☐	NEUTRIT			
☐	NEUTRON			#0036fe
☐	OMEGA-			
☐	OMEGAC0			
☐	OPTIPHOT			
☐	PHOTON			#FFFF00
■	PION+			#8B4513
■	PION-			#8B4513
☐	PIZERO			
☐	POSITRON			#FF00FF
■	PROTON			#ff0000
☐	RAY			
☐	SIGMA+			
☐	SIGMA-			
☐	SIGMAZER			
☐	TAU+			
☐	TAU-			
☐	TRITON			#B03060
☐	WWLOWNEU			

(3) **Follow the steps below in Flair**, to draw trajectories of interest in your geometry.

Step 1: Select the **Geometry** tab.

Step 2: (optional) **Clone an existing layer**, and rename it.

Step 3: **Add** a "Userdump" (active if filled black box).

Step 4: **Select the Userdump file** created by your FLUKA run (here, *_dump).

Step 5: **Select particles** of interest (active <-> filled black box).

Step 6: Double-click on **colour box** to update colour.

Step 7: **Select the layer** you were on at Step 2 in any window of your choice.

MGDRAW entries: ENDRAW

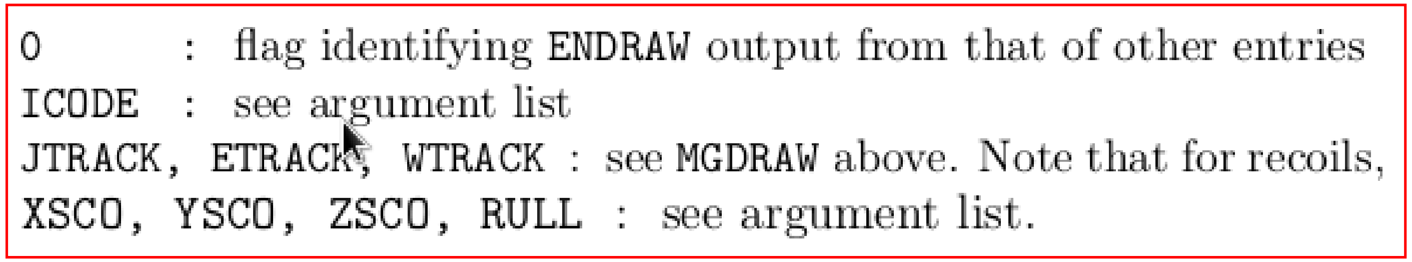
Local (point-like) energy deposition events

default implementation

```
 ENTRY ENDRAW (ICODE, MREG, RULL, XSCO, YSCO, ZSCO)
 ...
 WRITE (IODRAW)  0, ICODE, JTRACK,
&                SNGL(ETRACK), SNGL(WTRACK)
 WRITE (IODRAW)  SNGL (XSCO),
&                SNGL(YSCO), SNGL(ZSCO), SNGL(RULL)
 ...
 RETURN
```

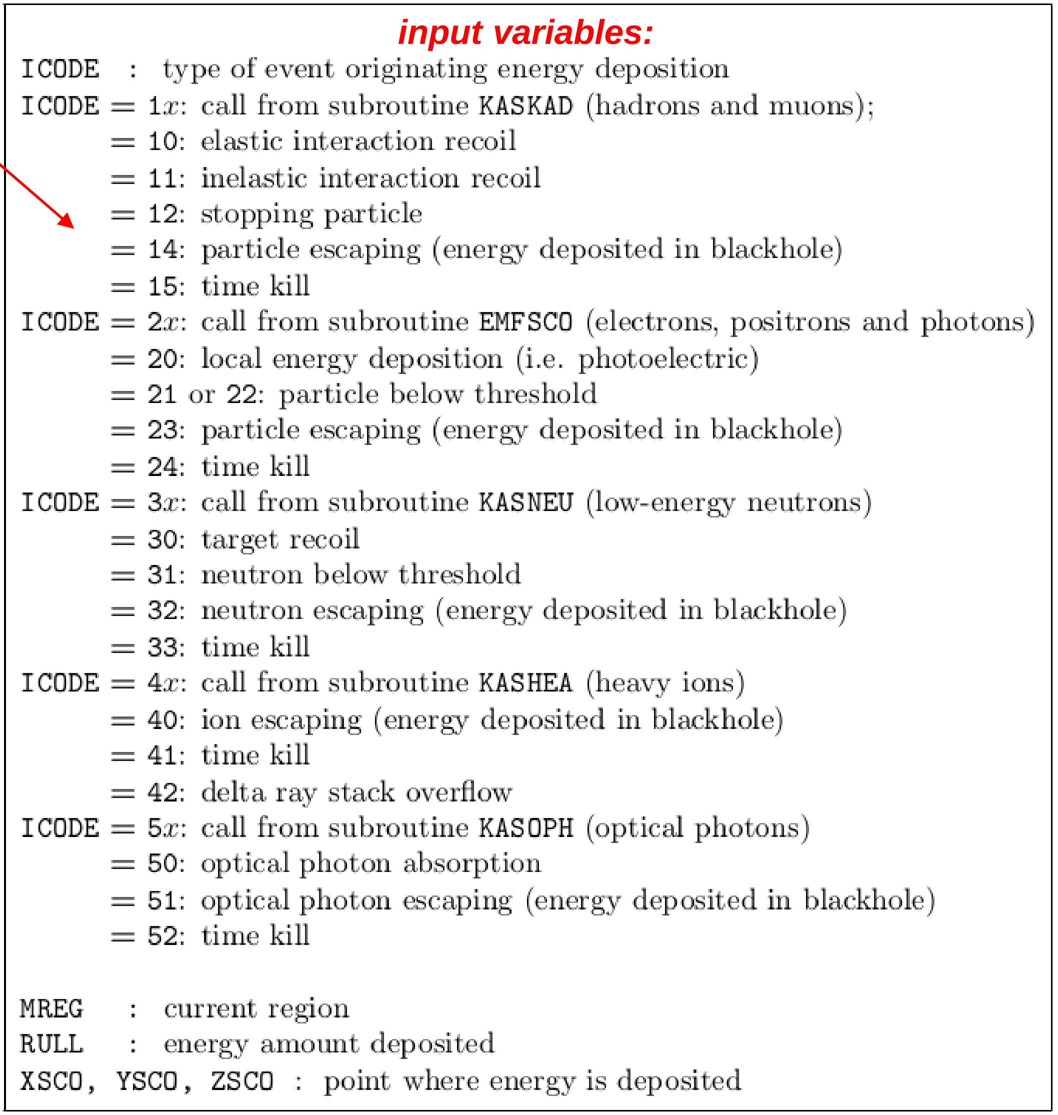
ENDRAW writes ***by default***, for **each energy deposition point**:

output variables:



0 : flag identifying ENDRAW output from that of other entries
ICODE : see argument list
JTRACK, ETRACK, WTRACK : see MGDRAW above. Note that for recoils,
XSCO, YSCO, ZSCO, RULL : see argument list.

input variables:



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)
= 15: time kill
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)
= 24: time kill
ICODE = 3x: call from subroutine KASNEU (low-energy neutrons)
= 30: target recoil
= 31: neutron below threshold
= 32: neutron escaping (energy deposited in blackhole)
= 33: time kill
ICODE = 4x: call from subroutine KASHEA (heavy ions)
= 40: ion escaping (energy deposited in blackhole)
= 41: time kill
= 42: delta ray stack overflow
ICODE = 5x: call from subroutine KASOPH (optical photons)
= 50: optical photon absorption
= 51: optical photon escaping (energy deposited in blackhole)
= 52: time kill

MREG : current region
RULL : energy amount deposited
XSCO, YSCO, ZSCO : point where energy is deposited

MGDRAW entries: SODRAW

Source particles

default implementation

```
 ENTRY SODRAW
...
 WRITE (IODRAW) -NCASE, NPFLKA,
& NSTMAX, SNGL (TKESUM), SNGL (WEIPRI)
...
! Default case
   ELSE
      WRITE (IODRAW) ( ILOFLK(I),
&           SNGL (TKEFLK(I)+AM(ILOFLK(I))),
&           SNGL (WTFLK(I)), SNGL (XFLK (I)),
&           SNGL (YFLK (I)), SNGL (ZFLK (I)),
&           SNGL (TXFLK(I)), SNGL (TYFLK(I)),
&           SNGL (TZFLK(I)), I = 1, NPFLKA )
 END IF

 RETURN
```

input variables: none

output variables:

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

SODRAW (when activated) writes ***by default***, for each source or beam particle:

output variables:

NPFLKA times: (all variables in COMMON FLKSTK)

flkstk.inc

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 entries: BXDRAW

Boundary-crossing events

default implementation (empty)

```
 ENTRY BXDRAW ( ICODE, MREG,
&                 NEWREG, XSCO, YSCO, ZSCO )
 RETURN
```

input variables:

```
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  :  coordinates of crossing point
```

BXDRAW (when activated) is **called at each boundary crossing**.
There is ***no default output***: any output must be supplied by the user.

MGDRAW ENTRIES: EEDRAW

End of events
(1 FLUKA event = 1 primary history)

default implementation (empty)

```
ENTRY EEDRAW ( ICODE )
RETURN
```

input variables:

```
ICODE  = -1: event not completed
       =  0: normal event termination
       =  4: stack overflow
```

EEDRAW (when activated) is **called at the end of each FLUKA event, or primary history**.
There is ***no default output***: any output must be supplied by the user.

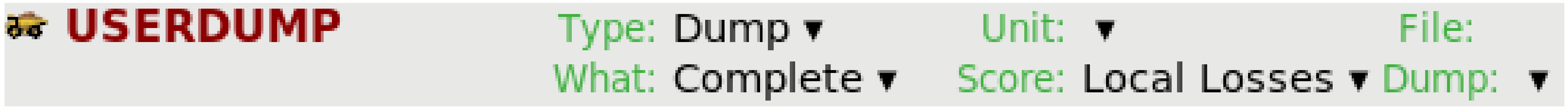
Example: Count particles crossing a surface for each primary event with BXDRAW and EEDRAW

We want to dump, *for each primary history*, the number of photons crossing a given boundary.
It is not possible to simply rely on default USRBDX: we would get the total number of photons crossing the boundary **for the entire run (and not per primary!)**.

BXDRAW allows us to study boundary-crossing, and EEDRAW to perform action at the end of each primary history.

We cannot use their default versions in mgdraw.f, **because they are empty**:
we need to **customize BXDRAW and EEDRAW to our needs**.

(1) **Activate BXDRAW and EEDRAW calls.**



USERDUMP Type: Dump ▾ Unit: ▾ File:
What: Complete ▾ Score: Local Losses ▾ Dump: ▾

```
* ..+....1....+....2....+....3....+....4....+....5....+....6....+....7..
USERDUMP             100.              3.
```

Example: Count particles crossing a surface for each primary event with BXDRAW and EEDRAW

user implementation

```
 ! Boundary crossing
 entry BXDRAW(icode, mreg, newreg, xsco, ysco, zsco)

 ! Initialization (only done once)
 if (first_run) then
    ! Converting region names into region numbers
    ! Region names must be padded with SPACEs up to 8 characters
    call GEON2R("TARG1   ", region_number_out, IERR )
    if (IERR /= 0) call FLABRT("bxdraw",
&                              "Failed region name conversion.")
    call GEON2R("TARG2   ", region_number_in, IERR )
    if (IERR /= 0) call FLABRT("bxdraw",
&                              "Failed region name conversion.")
    first_run = .false.
 end if

 ! Count optical photons only if they cross
 ! between the two specified regions
 if (MREG == region_number_out .and. NEWREG == region_number_in
&   .and. JTRACK == -1) then
    photon_counter = photon_counter + 1
 end if

 return
```

(2) **Modify BXDRAW and EEDRAW (in mgdraw.f)**

```
! Variables initialization
! (Put this inside MGDRAW subroutine, but
outside the BXDRAW / EEDRAW entries).
logical, save :: first_run = .true.
integer, save :: region_number_out,
region_number_in
integer, save :: photon_counter = 0
```

user implementation

```
! End of event (= primary history)
entry EEDRAW(icode)

! Write current primary number
! and number of counted photons
write(80, *) NCASE, photon_counter

! Reset photon counter for the next primary
photon_counter = 0

return
```

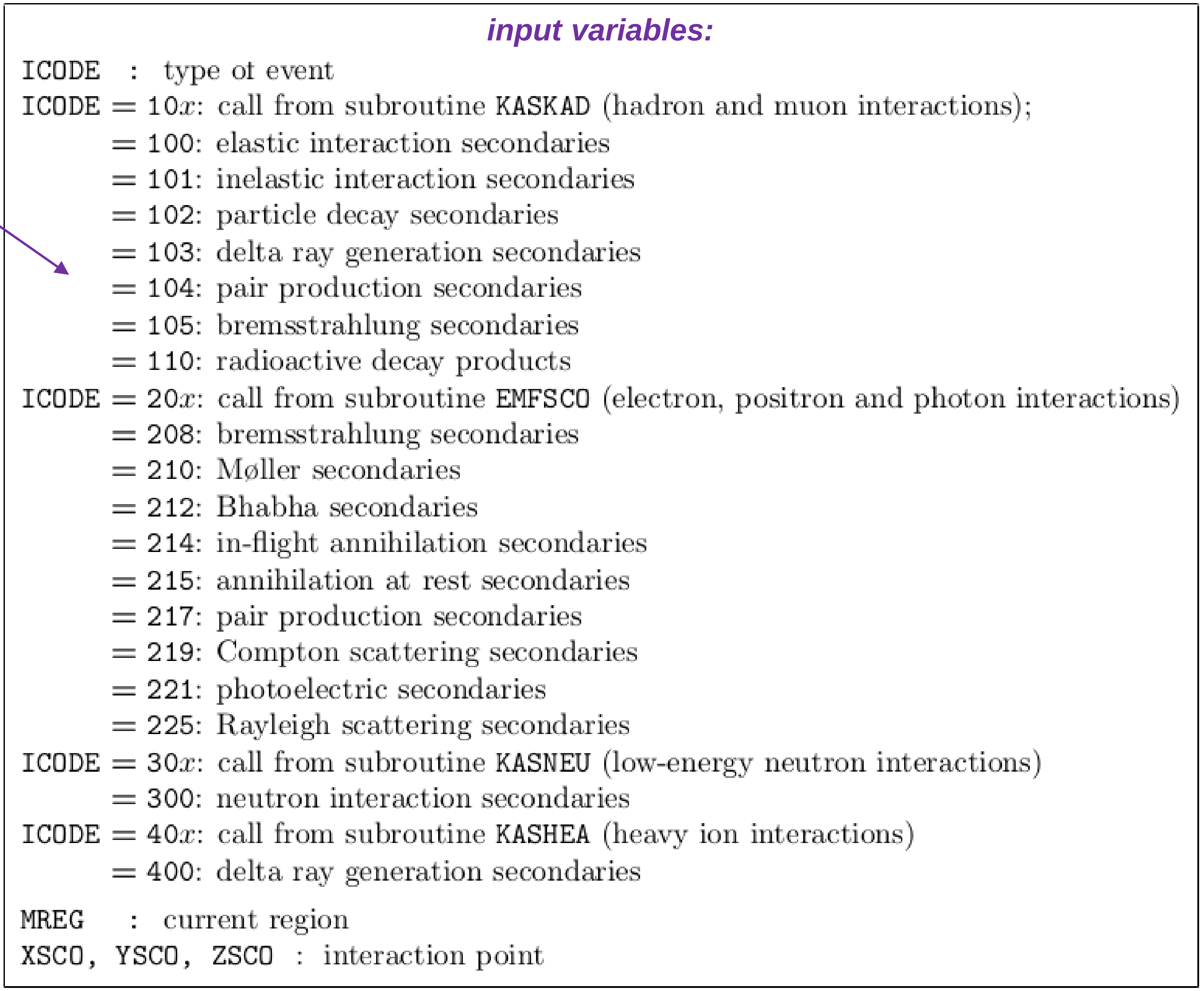
MGDRAW entries: USDRAW

User-defined dumps after interactions

default implementation (empty)

```
 ENTRY USDRAW ( ICODE, MREG,
&               XSCO, YSCO, ZSCO )
...
! No output by default:
 RETURN
```

USDRAW (when activated) is **called after each particle interaction**.
There is ***no default output***:
any output must be supplied by the user.



input variables:

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
= 110: radioactive decay products
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

USDRAW: get secondaries information

- **Secondaries properties** are available in **COMMON GENSTK** (indices 1 to NP: one per secondary). The surviving primary properties, if any, are also in GENSTK.
  Exception: delta rays produced by heavy ions. The properties of the single electron produced are available in **COMMON EMFSTK**, at index NP.
- **Heavy evaporation fragments**
  (deuterons, 3H, 3 He, α, with JTRACK ID equal respectively to -3, -4, -5, -6)
  + **fission/fragmentation products** generated in an inelastic interaction (with JTRACK = -7 to -12), are available in **COMMON FHEAVY**.
  Exception: heavy fragments from ion-ion interactions are in GENSTK.
- The properties of the **target nucleus** (IBTAR, ICHTAR...)
  + **residual nucleus**, if any (IBRES, ICHRES...)
  are in **COMMON RESNUC**.
  NB: This COMMON is not included in USDRAW by default.
- **EMF particles**: the code places them (temporarily!) in GENSTK, before calling USDRAW.

Example: print reaction final state with USDRAW

User implementation in mgdraw.f

```
  entry USDRAW(icode, mreg, xsco, ysco, zsco)

  if (icode .eq. 101 .and. JTRACK .eq. 1 .and. LTRACK .eq. 1) then

     write (90, *)
     write (90, *) 'PROJECTILE: id = ', JTRACK,
&         ', kE[GeV] = ', ETRACK, ', dirZ = ', CXTRCK

        write (90, *) 'Interaction id = ', ICODE,
&         ', in region = ', MREG

     write(90,*) 'NUMBER OF GENSTCK SECONDARIES = ', NP-NP0
     do jp = NP0+1, NP
        write(90,*) 'Sec: id = ', KPART(jp), ', kE[GeV] = ', TKI(jp)
     end do

     write(90,*) 'NUMBER OF HEAVY FRAGMENTS = ', NPHEAV
     do jp = 1, NPHEAV
        write(90,*) 'A = ', IBHEAV(KHEAVY(jp)),
&              ', Z = ', ICHEAV(KHEAVY(jp)), ', kE[GeV] = ', TKHEAV(jp)
     end do

     if (IBRES .gt. 0) then
        write(90,*) 'RESIDUAL NUCLEUS A = ', IBRES,
&              ', Z = ', ICRES, ', kE[GeV] = ', EKRES
     end if

  end if
  return
```

Filtering: inelastic interactions of a primary proton only

- Obviously can adapt to use case:
  - **reaction** code (ICODE)
  - **projectile** (JTRACK), **region** (MREG)
  - **primary history index** (NCASE)
  - **generation number** (LTRACK)...

Loop on GENSTCK secondaries

See `genstck.inc` for more properties

- **Start from NP0+1 to skip the primary when present**.
- NB: When the primary is not present, NP0=0.

Loop on heavy fragments `fheavy.inc`

Heavy residue `resnuc.inc`

End filtering

See Ex3 in exercise session.

Words of caution on MGDRAW use

- **When MGDRAW should be used with care:**

  When MGDRAW is used for *event-by-event scoring* (an event being here a ***full primary history***), it should NOT be used when non-physical transformations have been performed **within** the event:

  - **Biasing** is requested (non-analogue run).
  - **Groupwise low-energy neutron treatment (E<20 MeV)** is involved (unless one has a deep knowledge of the peculiarities of their transport and quantities, e.g. kerma, etc.).

- **Warning on output file size**

  - Be careful: **the MGDRAW output file can (very) quickly exceed several GBs**.
    This is because the number of MGDRAW calls is extensive: MGDRAW can be called after every particle step, or border crossing, or interaction etc.
    Exact file size is obviously dependent on your simulation and your MGDRAW implementation (if any).
    *Example: 450 GeV proton on a 400cm x 600cm Al target, default MGDRAW with "Traj&Cont losses", 1 cycle with **only 2 primaries**:*
    - *Dump file size is ~350 MB!*

Summary

- One should **always favour using the default FLUKA cards** rather than the user routines, and the **default user routines implementation** rather than a custom implementation.
- The MGDRAW routine is widely used to **access information on specific "events" of interest: it increases the level of granularity of the information accessible to the user**. The "event" can be a specific interaction, a boundary-crossing event, a local energy deposition event, etc.
- With the **default `mgdraw.f` implementation**, **MGDRAW** allows to dump information on trajectories, and continuous energy losses, **SODRAW** on source particles, **ENDRAW** on local (point-like) energy deposition events, while **BXDRAW, USDRAW, and EEDRAW are empty.**
- The user can **customize `mgdraw.f`** (potentially starting from `mgdraw_empty.f` template): he can tailor to his needs information dump in **MGDRAW** (trajectories, and continuous energy losses)**,** **SODRAW** (source particles), **ENDRAW** (point-like energy deposition events), **BXDRAW** (boundary-crossing events), **USDRAW** (interactions), **EEDRAW** (end of primary histories).
- MGDRAW calls need to be **activated by a FLUKA input card: USERDUMP.**
- Beware of **limitations on MGDRAW use.**

FLUKA