The FLUKA User Routines: how to tailor FLUKA to specific user's needs.

User programming in the FLUKA environment

7th FLUKA Course

NEA Paris, Sept.29-Oct.3, 2008

Why User Routines

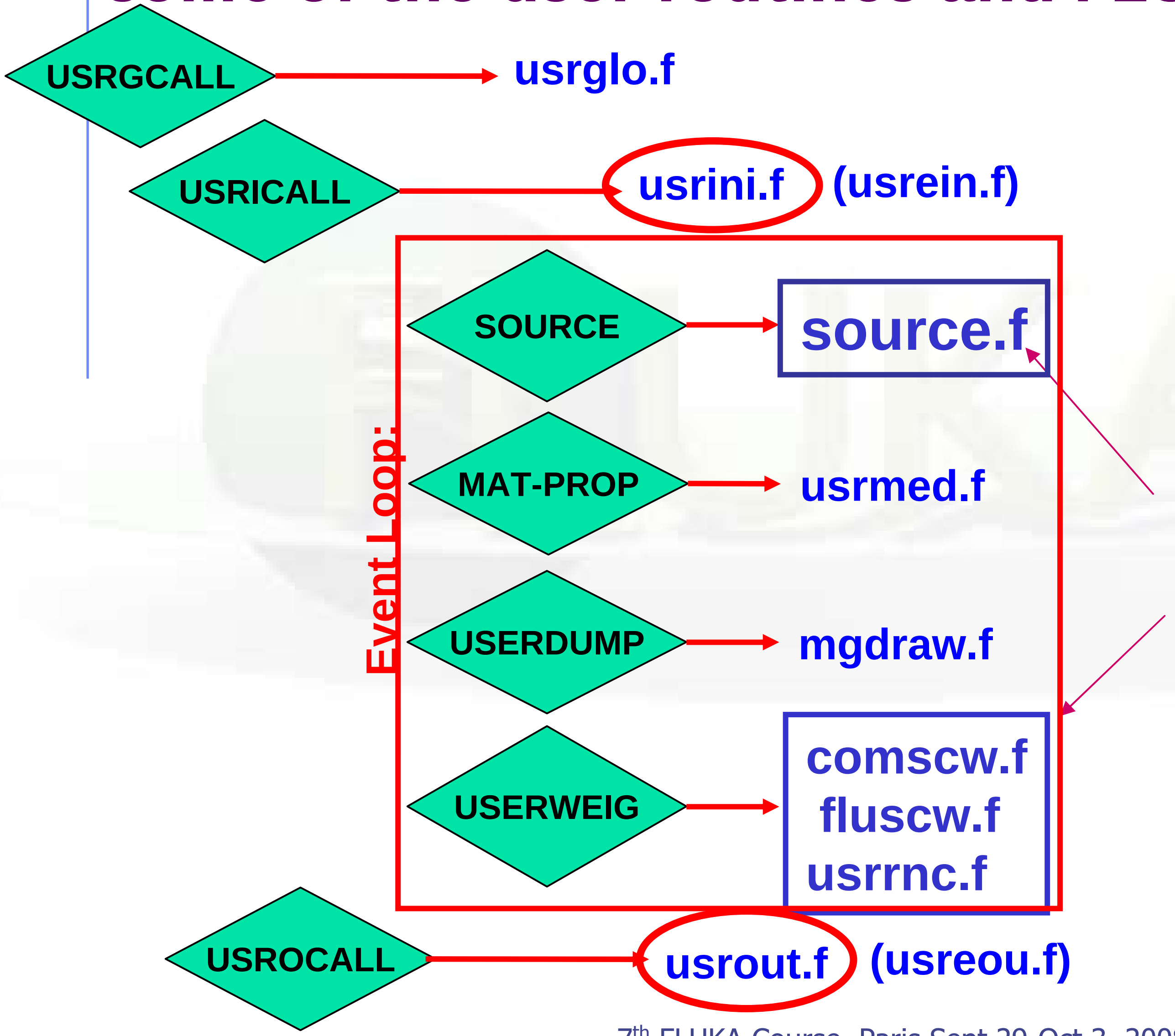
- Fluka offers a rich choice of options for scoring most quantities and for applying variance reduction techniques, without requiring the users to write a single line of code.
- However there are special cases where "ad-hoc" routines are unavoidable, because the required information cannot be obtained through standard options.
- A number of template of user routines (available in the usermvax directory) can be modified/activated by the user allow to fulfill non-standard tasks

What is available for the users

- The templates of all user routines are in the directory $FLUPRO/usermvax
- The include files containing the COMMON blocks are in the directory $FLUPRO/flukapro (see later)
- The compiling and linking scripts which are in the directory $FLUPRO/flutil

Flair can be used to edit, compile and link user routines in order to build a user-specific FLUKA executable

A first look at the correspondence between some of the user routines and FLUKA commands



Emphasis of this lecture is here

A possible classification in terms of their use (1)

User run control

- usrini.f
- usrein.f
- usrout.f
- usreou.f

Event generation, physics, kinematics

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

Properties of medium

- magfld.f
- usrmed.f

User global settings

- usrglo.f

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

To access (almost) everything

- mgdraw.f

See the relevant chap. of manual

Compiling and linking FLUKA user routines

- A FLUKA executable with user routines is in general application specific. It must be named and kept separately from the standard FLUKA
- Everything is managed today by FLAIR, however it is important to know the following details (managed automatically inside FLAIR):
- $FLUPRO/flutil/fff is the compiling script with the proper path to the INCLUDE subdirectory and the required compiler (g77) options

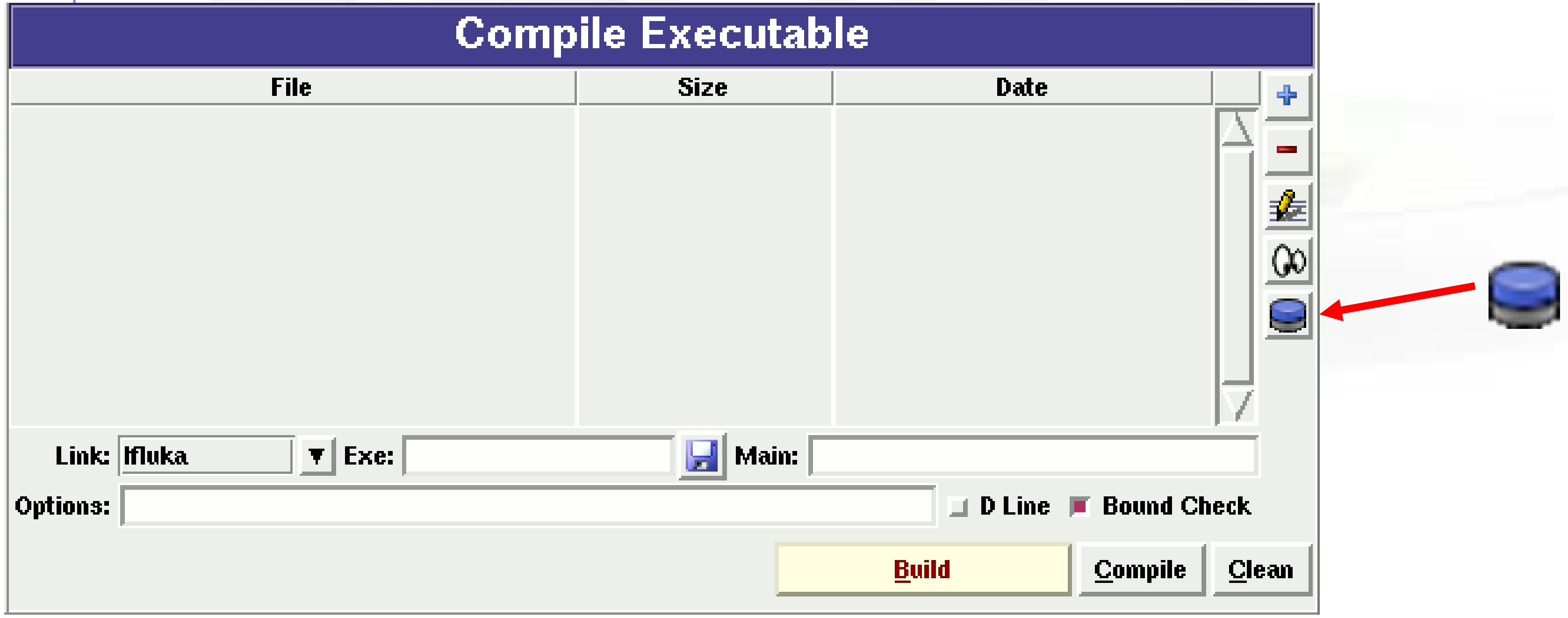
  Example: $FLUPRO/flutil/fff usrini.f generates usrini.o

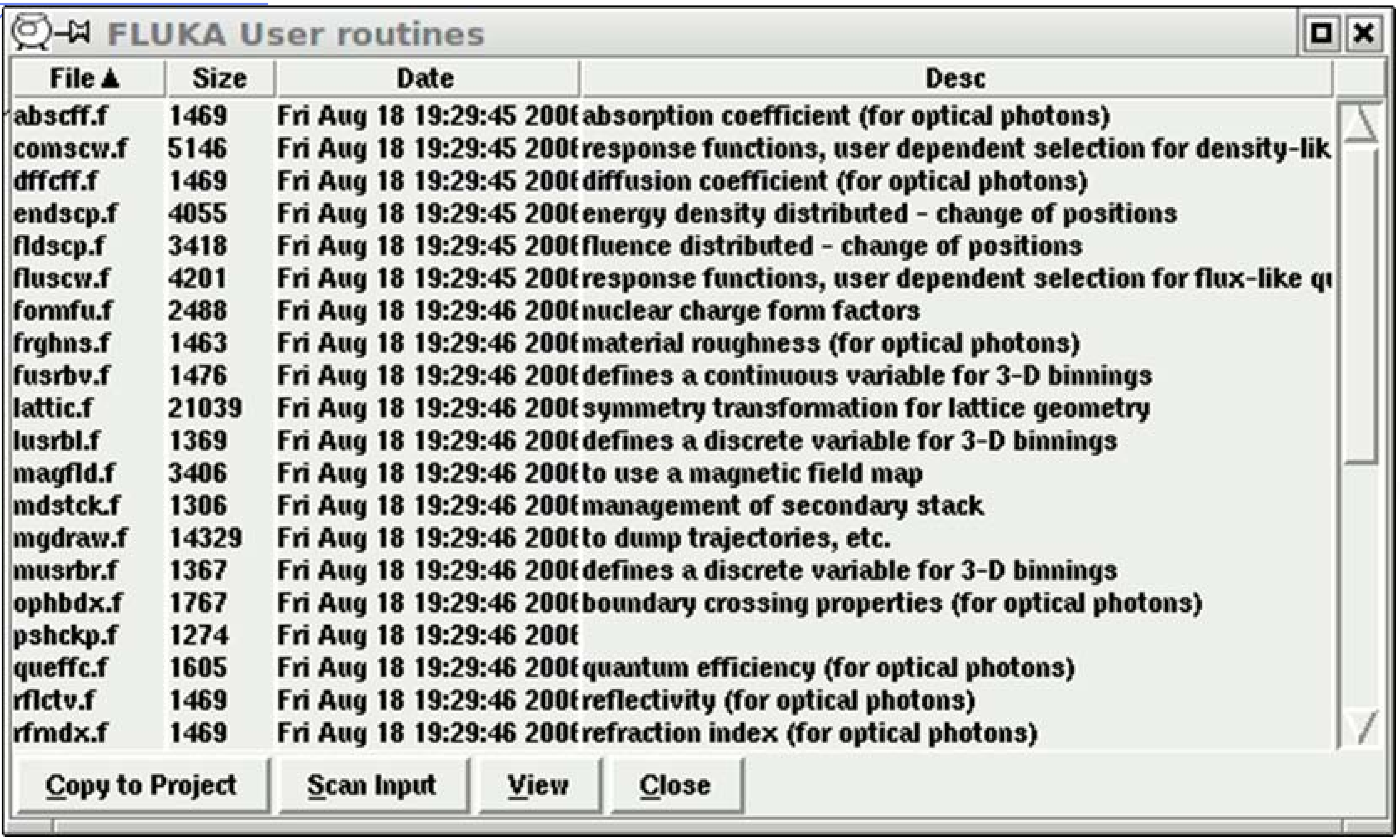
  then $FLUPRO/flutil/lfluka –m fluka –o flukamy usrini.o will perform the

  proper linking generating the executable here called flukamy
- Tip: $FLUPRO/flutil/lfluka –m fluka –o flukamy usrini.f will automatically call $FLUPRO/flutil/fff

Help by FLAIR

- FLAIR has a button in the Compile frame which allows to scan the input file for possible cards that require the use of user routines
- It allows to copy the template routine from $FLUPRO/usermvax to the project directory




FLUKA User routines

File ▲	Size	Date	Desc
abscff.f	1469	Fri Aug 18 19:29:45 200	absorption coefficient (for optical photons)
comscw.f	5146	Fri Aug 18 19:29:45 200	response functions, user dependent selection for density-lik
dffcff.f	1469	Fri Aug 18 19:29:45 200	diffusion coefficient (for optical photons)
endscp.f	4055	Fri Aug 18 19:29:45 200	energy density distributed - change of positions
fldscp.f	3418	Fri Aug 18 19:29:45 200	fluence distributed - change of positions
fluscw.f	4201	Fri Aug 18 19:29:45 200	response functions, user dependent selection for flux-like q
formfu.f	2488	Fri Aug 18 19:29:46 200	nuclear charge form factors
frghns.f	1463	Fri Aug 18 19:29:46 200	material roughness (for optical photons)
fusrbv.f	1476	Fri Aug 18 19:29:46 200	defines a continuous variable for 3-D binnings
lattic.f	21039	Fri Aug 18 19:29:46 200	symmetry transformation for lattice geometry
lusrbl.f	1369	Fri Aug 18 19:29:46 200	defines a discrete variable for 3-D binnings
magfld.f	3406	Fri Aug 18 19:29:46 200	to use a magnetic field map
mdstck.f	1306	Fri Aug 18 19:29:46 200	management of secondary stack
mgdraw.f	14329	Fri Aug 18 19:29:46 200	to dump trajectories, etc.
musrbr.f	1367	Fri Aug 18 19:29:46 200	defines a discrete variable for 3-D binnings
ophbdx.f	1767	Fri Aug 18 19:29:46 200	boundary crossing properties (for optical photons)
pshckp.f	1274	Fri Aug 18 19:29:46 200	
queffc.f	1605	Fri Aug 18 19:29:46 200	quantum efficiency (for optical photons)
rflctv.f	1469	Fri Aug 18 19:29:46 200	reflectivity (for optical photons)
rfrndx.f	1469	Fri Aug 18 19:29:46 200	refraction index (for optical photons)

Basics about FLUKA routines/functions

- Written in Fortran 77
- Double Precision everywhere, except for variables beginning with a letter within (i-n)
- Common blocks are in files which are loaded by INCLUDE statement
- All include files are in $FLUPRO/flukapro
- Each routine must start with the following includes/common blocks:

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

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

- Users may add their own common(s) which may reside in different places

Basic FLUKA Include Files

DBLPRC:	included in all routines of Fluka, contains (as PARAMETERS) the most common physical and mathematical constants and the declaration IMPLICIT DOUBLE PRECISION (A-H,O-Z)
DIMPAR:	dimensions of the most important arrays
IOUNIT:	logical input and output unit numbers (FLUKA uses those from 1 to 19, ***they must be considered as reserved***)

- Users are encouraged to adhere to the "Fluka style" by using systematically double precision (except for calling external single precision scoring packages), and to use constants defined in this file for maximum accuracy and consistency
- Important: take some time to study the content of DBLPRC

Some important COMMON blocks in short (1)

BEAMCM:	beam properties of primary (BEAM and BEAMPOS)
CASLIM:	number of primary particles followed
EMFSTK:	particle stack for electrons and photons
SOURCM:	user variables and information for a user-written source
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
SOUEVT:	variables describing the source event
TRACKR:	variables concerning the properties of transported particle (track) at run time
PAPROP:	particle properties (masses, charges, mean lives…)
SCOHLP:	variables concerning the current estimator type

Converting Names↔Number

- FLUKA converts all Names given in the input file to Numbers: all the arguments that you will find in user routines are numeric

To get the number starting from a region name

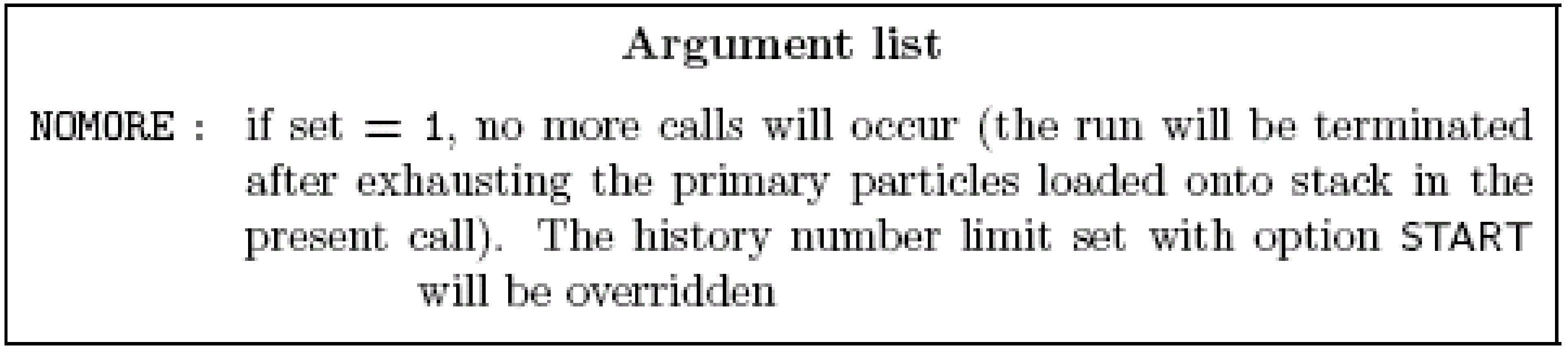
```
CALL GEON2R ( REGNAM, NREG, IERR )
*     Input variable:
*        Regnam   = region name   (CHAR*8)
*
*     Output variables:
*        Nreg     = region number
*        Ierr     = error code (0 on success, 1 on failure)
```

Similar routines for lattice geometry

To get the name of a region when you know the number:

```
CALL GEOR2N ( NREG, REGNAM, IERR )
*     Input variable:
*        Nreg    = region number
*
*     Output variables:
*        Regname  = region name  (CHAR*8)
*        Ierr     = error code (0 on success, 1 on failure)
```

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

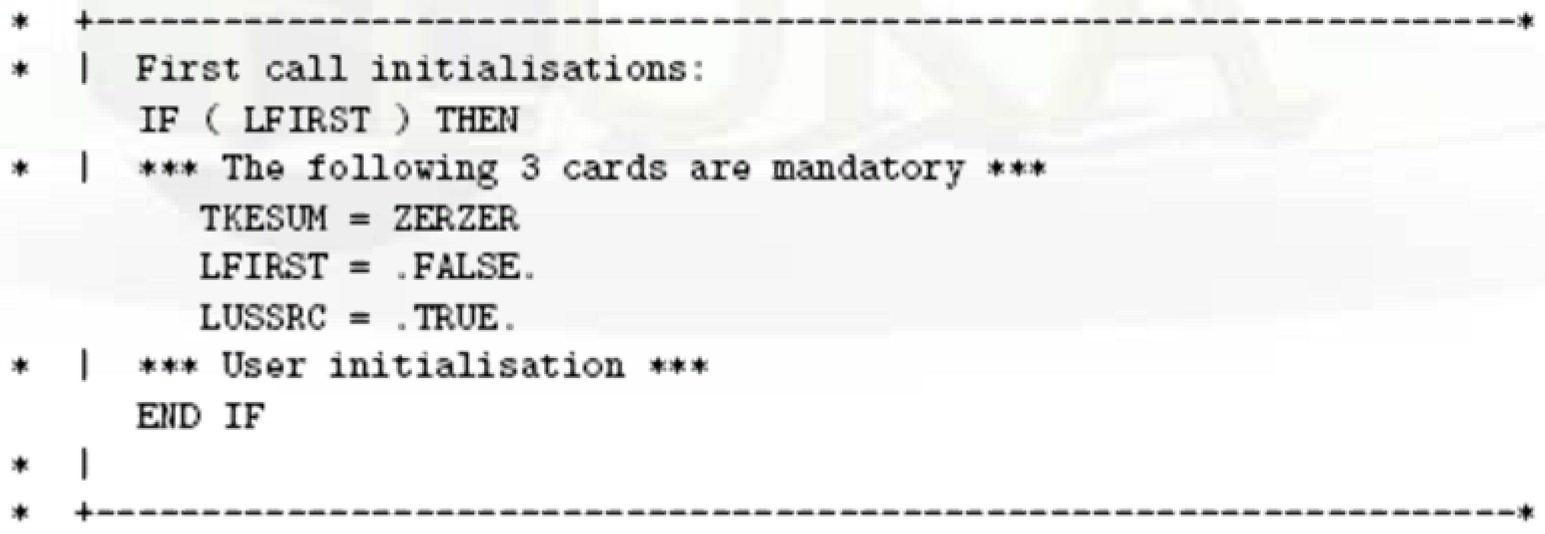
Subroutine SOURCE is probably the most frequently used user routine. It is activated by option SOURCE and is used to sample primary particle properties from distributions (in space, energy, time, direction or mixture of particles) which cannot be described with the BEAM, BEAMPOS and BEAMAXES cards. At each call, one (or more) particle(s) 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 18 numerical values (WHASOU(1),(2). . . (18)) 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 first time 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 particles at each call: for each particle loaded the pointer must be increased by 1

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

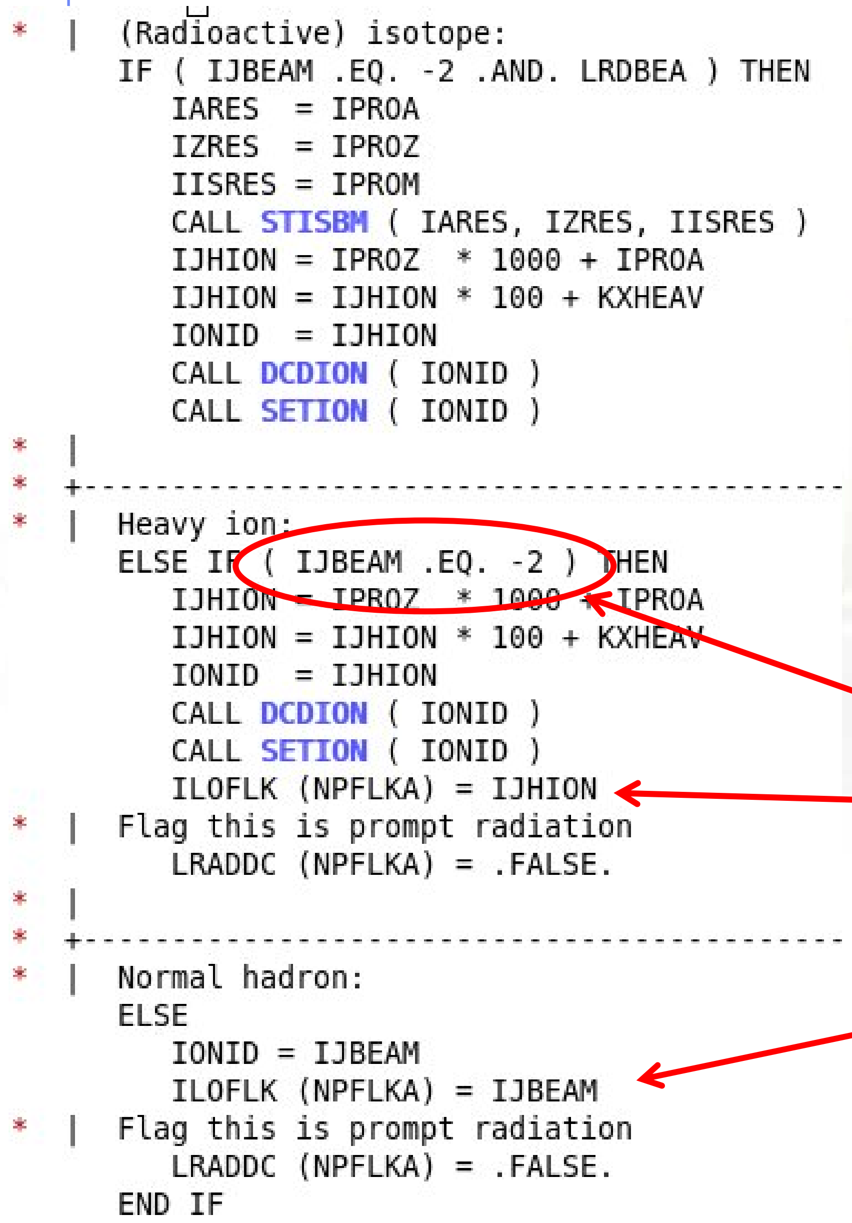
weight of the particle (values different from 1 → biased source, advanced users)

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

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

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

Using source: setting the particle id



```
*  |  (Radioactive) isotope:
      IF ( IJBEAM .EQ. -2 .AND. LRDBEA ) THEN
         IARES  = IPROA
         IZRES  = IPROZ
         IISRES = IPROM
         CALL STISBM ( IARES, IZRES, IISRES )
         IJHION = IPROZ  * 1000 + IPROA
         IJHION = IJHION * 100 + KXHEAV
         IONID  = IJHION
         CALL DCDION ( IONID )
         CALL SETION ( IONID )
*  |
*  +-------------------------------------------
*  |  Heavy ion:
      ELSE IF ( IJBEAM .EQ. -2 ) THEN
         IJHION = IPROZ  * 1000 + IPROA
         IJHION = IJHION * 100 + KXHEAV
         IONID  = IJHION
         CALL DCDION ( IONID )
         CALL SETION ( IONID )
         ILOFLK (NPFLKA) = IJHION
*  |  Flag this is prompt radiation
         LRADDC (NPFLKA) = .FALSE.
*  |
*  +-------------------------------------------
*  |  Normal hadron:
      ELSE
         IONID = IJBEAM
         ILOFLK (NPFLKA) = IJBEAM
*  |  Flag this is prompt radiation
         LRADDC (NPFLKA) = .FALSE.
      END IF
```

The template sets the type of particle equal to the one defined by the BEAM card (plus HI-PROPE if used).

Whichever valid particle id can be set inside the source (may be different event by event)

Heavy ion

“Normal” particle

Using source: assigning momentum energy

In the template routine, the momentum is taken from the BEAM option (PBEAM, in COMMON BEAMCM, which contains all values defined by options BEAM and BEAMPOS)

```
    PMOFLK (NPFLKA) = PBEAM
* Kinetic energy of the particle (GeV)
    TKEFLK (NPFLKA) = SQRT ( PBEAM**2 + AM (IONID)**2 ) - AM (IONID)
```

The user can select a momentum from a spectrum or a file

Alternatively the user can sample/assign the kinetic energy and derive the momentum. Be coherent!

```
     TKEFLK (NPFLKA) = ENSAMP
     PMOFLK (NPFLKA) = SQRT ( TKEFLK (NPFLKA) * ( TKEFLK (NPFLKA)
    &                         + TWOTWO * AM (IONID) ) )
```

Even when using SOURCE, the BEAM card remains mandatory. The momenta (or energies) assigned in SOURCE can never be larger than the momentum/energy set in BEAM (it is the one used during initialization)!!

Using source: setting position and direction

Direction cosines assignment:

```
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
```

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
```

Be careful to ensure the cosine proper normalization within machine accuracy!!! Ie...

```
      TNORM = SQRT ( TXFLK(NPFLKA)**2 + TYFLK(NPFLKA)**2 + TZFLK(NPFLKA)
     $     **2)
      TXFLK  (NPFLKA) = TXFLK(NPFLKA) / TNORM
      TYFLK  (NPFLKA) = TYFLK(NPFLKA) / TNORM
      TZFLK  (NPFLKA) = TZFLK(NPFLKA) / TNORM
```

Using source (continues...)

The following lines can remain as they are

```
AGESTK (NPFLKA) = +ZERZER         !  Particle age is zero by default
AKNSHR (NPFLKA) = -TWOTWO         !  Resets the Kshort component of
                                  !  K0/K0bar. Usually not to be changed.
IGROUP (NPFLKA) = 0               !  Group number for low-energy
                                  !  neutrons: if set to 0, the program
                                  !  derives it from the kinetic energy
```

Don’t change the lines below:

```
* 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
        .......
      DFNEAR (NPFLKA) = +ZERZER     !  Resetting the distance to the
                                    !  nearest boundary
* ... to this point: don't change anything
```

At the end, source makes a copy into SOEVSV of the generated particles in case this info is required by some user routine at scoring stage

Using the FLUKA Random Number Generator in user routines

Fundamental for SOURCE!!! No other external random generators must be used, otherwise the history reproducibility will be lost

- … = FLRNDM (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 SINT and COST, sine and cosine of a random azimuthal angle

SINT**2 + COST**2 = 1.D+00

- CALL RACO (TXX, TYY, TZZ)

returns a random 3D direction (TXX, TYY, TZZ) such that:

TXX**2 + TYY**2 + TZZ**2 = 1.D+00

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)

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)

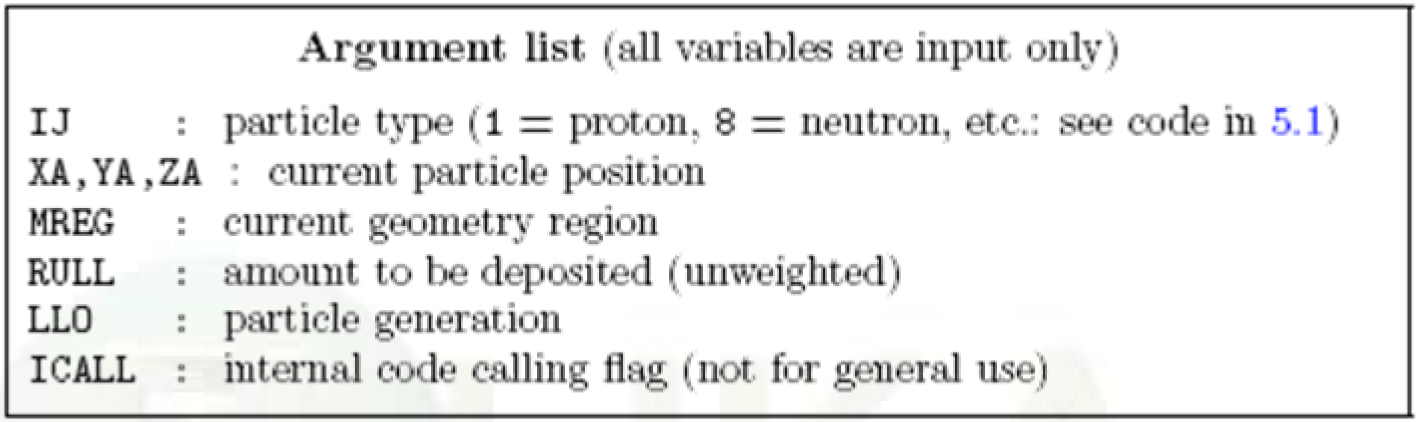
CALL SFLOOD (XXX, YYY, ZZZ, UXXX, VYYY, WZZZ)

returns in XXX, YYY, ZZZ a random position ON the surface of a sphere of radius 1 and centre 0 (multiply XXX, YYY, ZZZ by the actual radius and add the centre coordinates) and UXXX, VYYY, WZZZ are random cosines distributed so as to generate a uniform and isotropic fluence inside the sphere numerically given by

$$1/(\pi R^2)$$

R being the sphere radius.

comscw (weighting energy deposition or star production)



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)

Activated by option USERWEIG with WHAT(6) > 0.0. Energy and stars obtained via SCORE, USRBIN and 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 according to 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 $\rightarrow$ Energy density binning   ISCRNG = 2 $\rightarrow$ Star density binning
ISCRNG = 3 $\rightarrow$ Residual nuclei scoring

comscw (continues...)

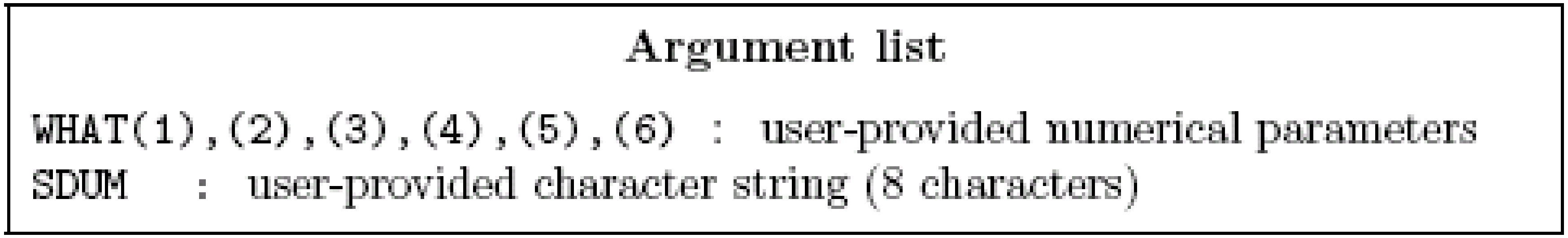
The binning/detector number is given by JSCRNG (in SCOHLP) and is printed in output between the estimator type and the detector name.

Note that a 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.

usrini (USeR INItialization)



Argument list

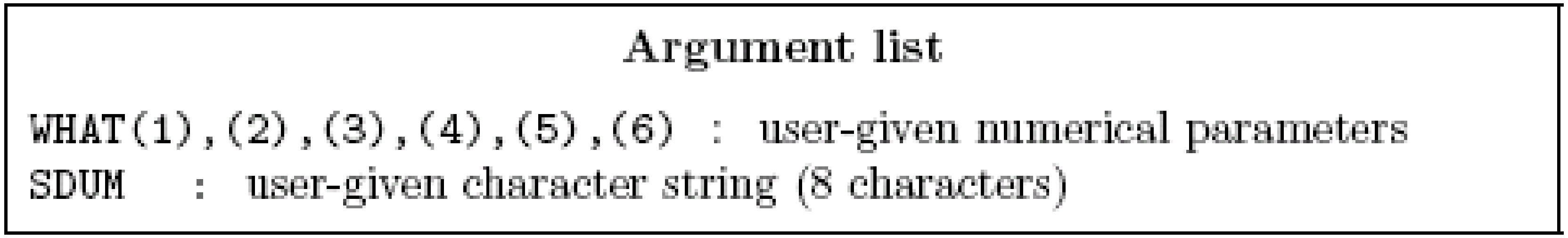
`WHAT(1),(2),(3),(4),(5),(6)`	:	user-provided numerical parameters
`SDUM`	:	user-provided character string (8 characters)

Subroutine USRINI is called every time a USRICALL card is read in the input stream, before particle showering starts. Useful for initialisation: ie reading and manipulating data from one or more files, calling other private routines, etc.
The calling parameters can be used by the user to pass variables/flags to the routine.

usrein (USeR Event INitialization)

Subroutine USREIN is called just before the start of an event. 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; there can be more if the user routine SOURCE is used to load particles into stack. USREIN is always called: the default version of USREIN does nothing.

usrout (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 the input stream. It is used for user-written output in addition to the standard one provided by default. The calling parameters can be used by the user to pass variables/flags to the routine.

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.

USREOU is always called: the default version of USREOU does nothing. The user can plug in any kind of event analysis, output, etc.

Mathematical library in FLUKA

- FLUKA contains many mathematical routines of general utility, so in general it should not be necessary to call external mathematical libraries:

flgaus:	Gaussian adaptative integration
simpsn:	Simpson integration
gamfun:	Gamma fuction
radcub:	Real solutions of 3rd order algebric equation
flgndr:	Legendre polinomials
yinter, finter, d..intp:	interpolation routines
rordin, rordde:	Sorting of vector values

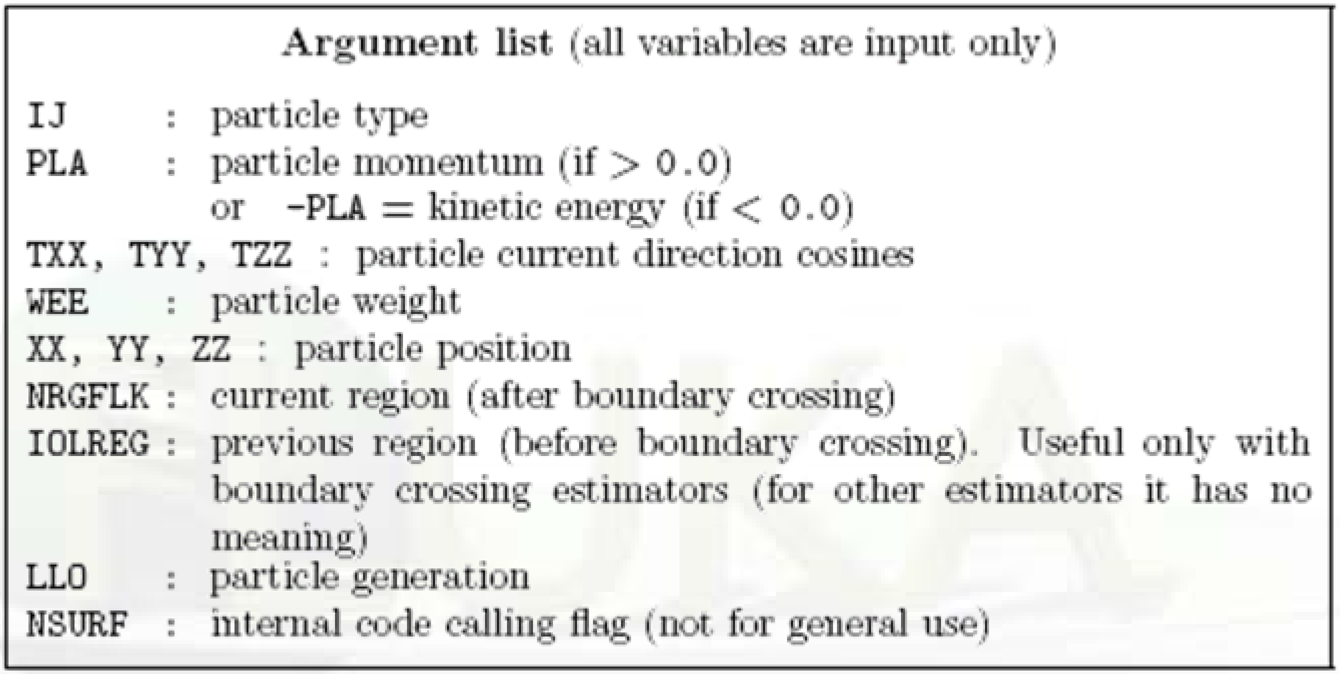
....

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

For users who access the FLUKA source: they are in mathmvax directory

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

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...)

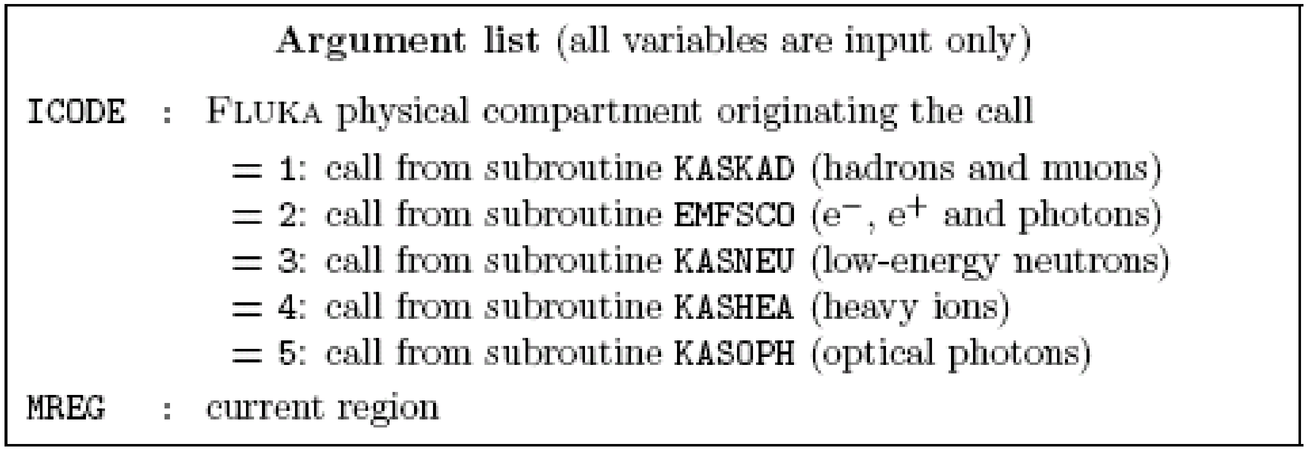
The user can implement any desired logic according to the argument list (particle type, energy, direction, weight, position, region, boundary, particle generation), or information available in COMMON SCOHLP (binning or detector number, estimator type). The estimator type is given by the flag ISCRNG (in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

A very special user routine: mgdraw.f

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)

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

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	called at each step, for trajectory drawing and dE/dx energy deposition events,
ENDRAW	for recording of point energy deposition events,
SODRAW	for recording of source events

mgdraw (continues...)

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).

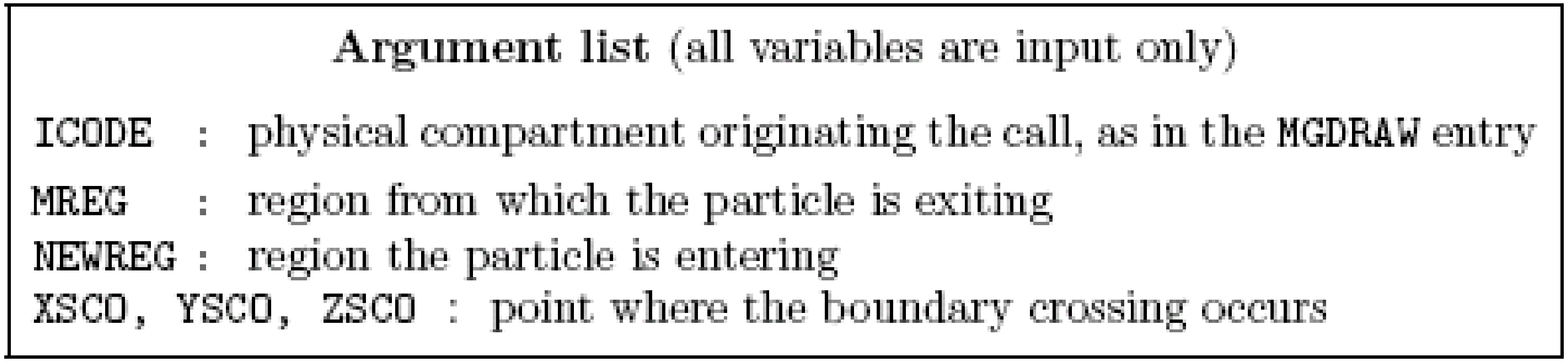
mgdraw: the MGDRAW entry

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

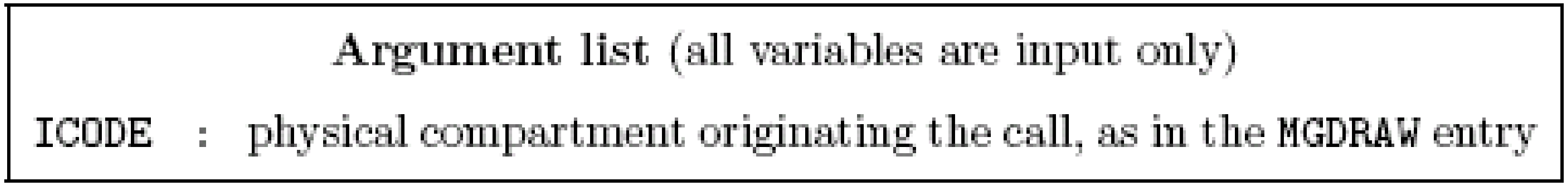


Argument list (all variables are input only)

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

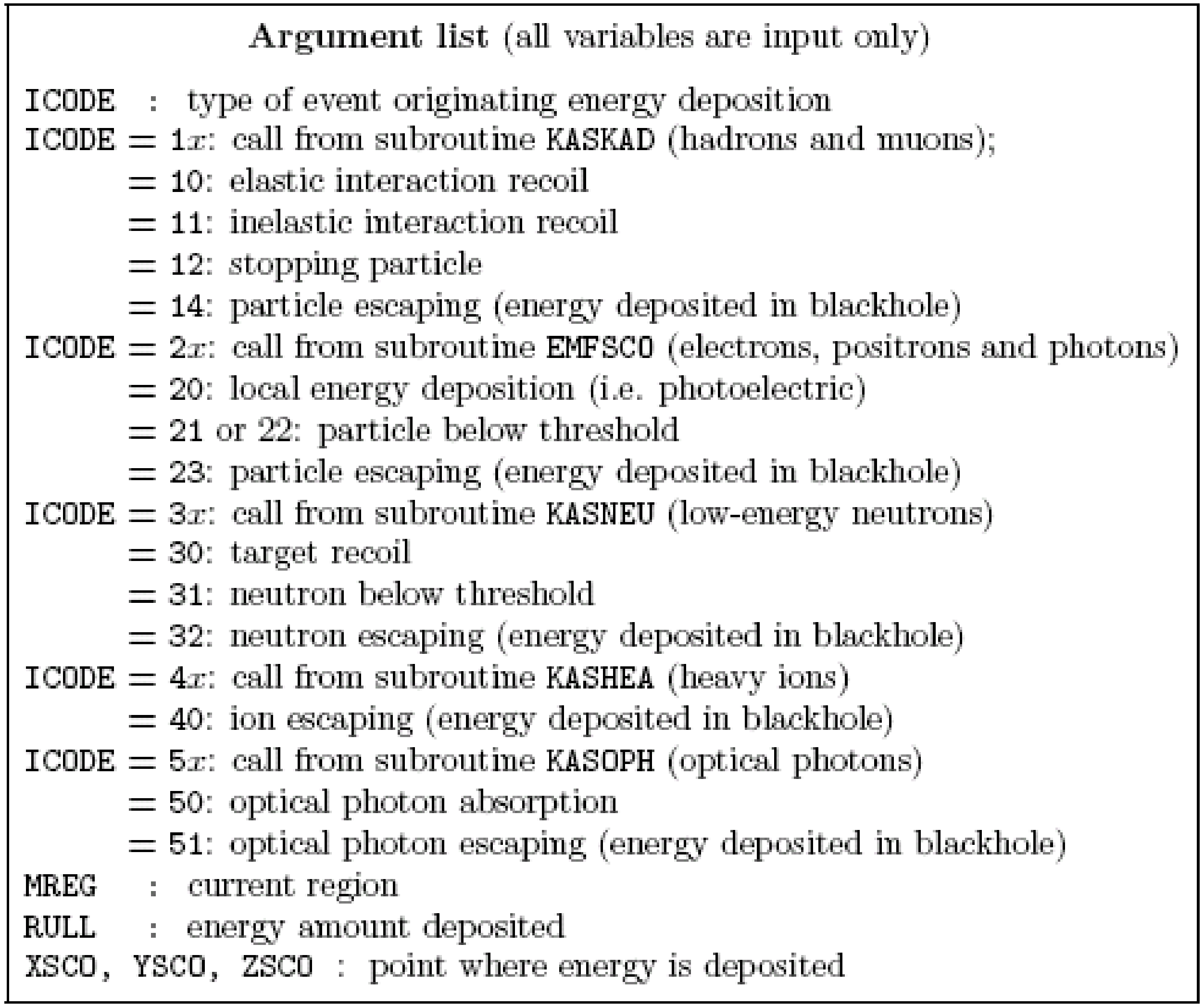


Argument list (all variables are input only)		
ICODE	:	physical compartment originating the call, as in the MGDRAW entry

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

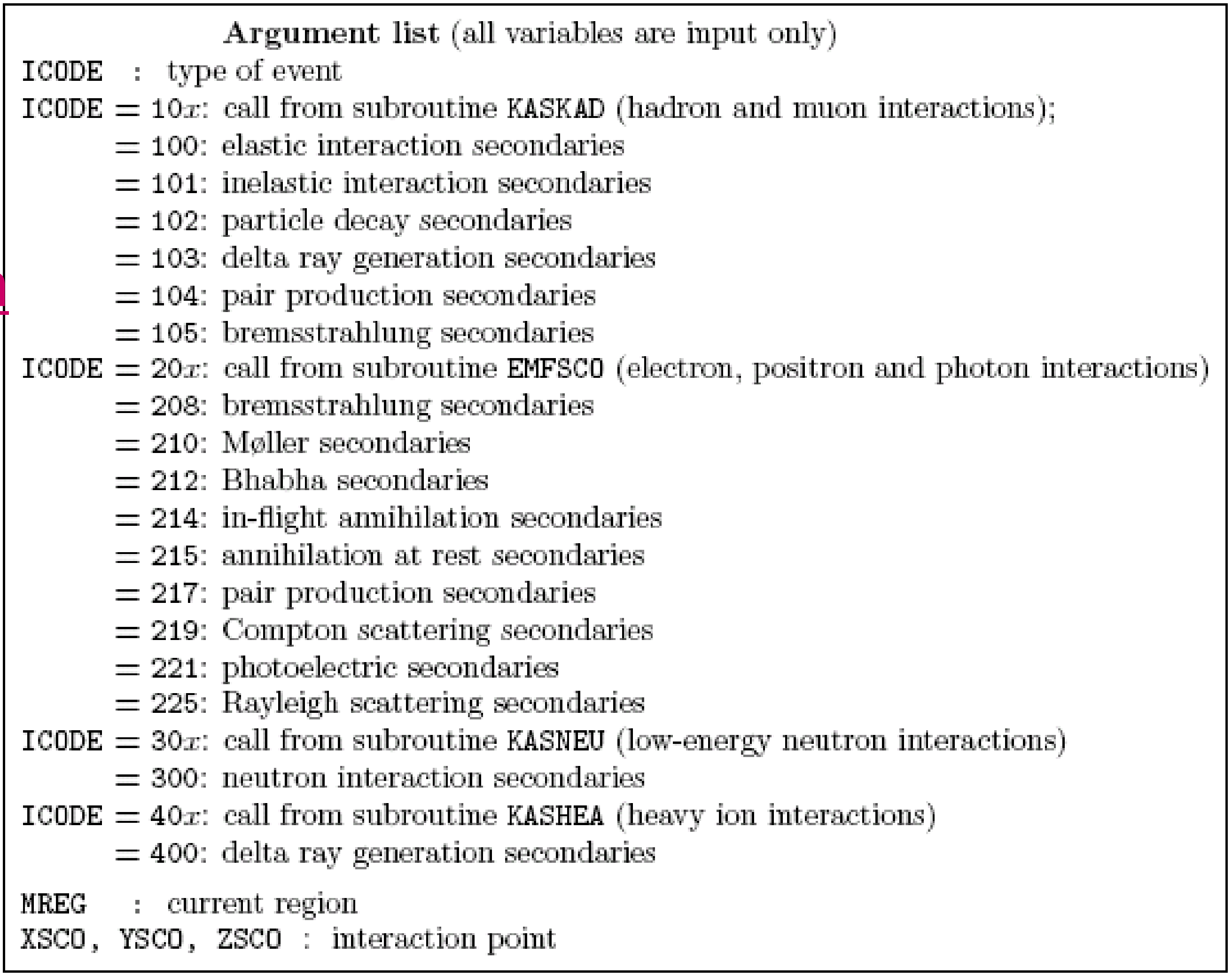
SODRAW writes by default, for each source or beam particle:

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

When mgdraw should better not be used

- When biasing is requested
- Whenever low-energy neutrons ($E<20$ MeV) are used, unless one has a deep knowledge of the peculiarities of their transport (ie kerma, etc)

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

END