Source routines

An introduction to a new approach to source routines

Why user routines?

- FLUKA offers plenty of built-in tools to define primary beams and estimate quantities
- Sometime these are not enough
- There is the need to write some dedicated code: a “User Routine”
- URs are beyond the scope of this course because of intrinsic difficulties
- Nevertheless, we have started an effort to make URs more user-friendly
- We want to introduce here the first effort in this direction:

  a modernized version of the **source routine**
- Why the source routine first? Built-in options allow to sample from a limited number

  of distribution and not from histograms. This is an effort to overcome this limitation

The “old” source routine

- Scary for beginners, limited documentation
- Use of **IMPLICIT** and **FORTRAN77** naming convention (see later)

```
*
*=== source ============================================================*
*
      SUBROUTINE SOURCE ( NOMORE )

      INCLUDE 'dblprc.inc'
      INCLUDE 'dimpar.inc'
      INCLUDE 'iounit.inc'
*
*----------------------------------------------------------------------*
*                                                                      *
*     Copyright (C) 2003-2019:  CERN & INFN                            *
*     All Rights Reserved.                                             *
*                                                                      *
*     New source for FLUKA9x-FLUKA20xy:                                *
*                                                                      *
*     Created on 07 January 1990   by    Alfredo Ferrari & Paola Sala  *
*                                                Infn - Milan          *
*                                                                      *
*  This is just an example of a possible user written source routine.  *
*  note that the beam card still has some meaning - in the scoring the *
*  maximum momentum used in deciding the binning is taken from the     *
*  beam momentum.  Other beam card parameters are obsolete.            *
*                                                                      *
*       Output variables:                                              *
*                                                                      *
*              Nomore = if > 0 the run will be terminated              *
*                                                                      *
*----------------------------------------------------------------------*
*
      INCLUDE 'beamcm.inc'
      INCLUDE 'fheavy.inc'
      INCLUDE 'flkstk.inc'
      INCLUDE 'ioiocm.inc'
      INCLUDE 'ltclcm.inc'
      INCLUDE 'paprop.inc'
      INCLUDE 'sourcm.inc'
      INCLUDE 'sumcou.inc'
*
      LOGICAL LFIRST, LISNUT
*
      SAVE LFIRST
      DATA LFIRST / .TRUE. /
*  Statement function:
      LISNUT (IJ) = INDEX ( PRNAME (IJ), 'NEUTRI' ) .GT. 0
*======================================================================*
*                                                                      *
*                BASIC VERSION                                         *
*                                                                      *
*======================================================================*
      NOMORE = 0
*  +-------------------------------------------------------------------*
*  |  First call initializations:
      IF ( LFIRST ) THEN
*  |  *** The following 3 cards are mandatory ***
         TKESUM = ZERZER
         LFIRST = .FALSE.
         LUSSRC = .TRUE.
*  |  *** User initialization ***
      END IF
*  |
*  +-------------------------------------------------------------------*
```

```
*  Push one source particle to the stack. Note that you could as well
*  push many but this way we reserve a maximum amount of space in the
*  stack for the secondaries to be generated
*  Npflka is the stack counter: of course any time source is called it
*  must be =0
      NPFLKA = NPFLKA + 1
*  Wt is the weight of the particle
      WTFLK  (NPFLKA) = ONEONE
      WEIPRI = WEIPRI + WTFLK (NPFLKA)
*  Particle type (1=proton.....). Ijbeam is the type set by the BEAM
*  card
*  +-------------------------------------------------------------------*
*  |  (Radioactive) isotope:
      IF ( IJBEAM .EQ. -2 .AND. LRDBEA ) THEN
         IARES  = IPROA
         IZRES  = IPROZ
         IISRES = IPROM
         CALL STISBM ( IARES, IZRES, IISRES )
         IJHION = IPROM  * 100000 + MOD ( IPROZ, 100 ) * 1000 + IPROA
         IJHION = IJHION * 100    + KXHEAV
         IONID  = IJHION
         CALL DCDION ( IONID )
         CALL SETION ( IONID )
         LFRPHN (NPFLKA) = .FALSE.
*  |
*  +-------------------------------------------------------------------*
*  |  Heavy ion:
      ELSE IF ( IJBEAM .EQ. -2 ) THEN
         IJHION = IPROM  * 100000 + MOD ( IPROZ, 100 ) * 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.
*  |  Group number for "low" energy neutrons, set to 0 anyway
         IGROUP (NPFLKA) = 0
*  |  Parent radioactive isotope:
         IRDAZM (NPFLKA) = 0
*  |  Particle age (s)
         AGESTK (NPFLKA) = +ZERZER
*  |  Kinetic energy of the particle (GeV)
         TKEFLK (NPFLKA) = SQRT ( PBEAM**2 + AM (IONID)**2 )
     &                   - AM (IONID)
*  |  Particle momentum
         PMOFLK (NPFLKA) = PBEAM
         PMOFLK (NPFLKA) = SQRT ( TKEFLK (NPFLKA) * ( TKEFLK (NPFLKA)
*    &                           + TWOTWO * AM (IONID) ) )
         LFRPHN (NPFLKA) = .FALSE.
*  |
*  +-------------------------------------------------------------------*
*  |  Normal hadron:
      ELSE
         IONID = IJBEAM
         ILOFLK (NPFLKA) = IJBEAM
*  |  Flag this is prompt radiation
         LRADDC (NPFLKA) = .FALSE.
*  |  Group number for "low" energy neutrons, set to 0 anyway
         IGROUP (NPFLKA) = 0
*  |  Parent radioactive isotope:
         IRDAZM (NPFLKA) = 0
*  |  Particle age (s)
```

```
         AGESTK (NPFLKA) = +ZERZER
*  |  Kinetic energy of the particle (GeV)
         TKEFLK (NPFLKA) = SQRT ( PBEAM**2 + AM (IONID)**2 )
     &                   - AM (IONID)
*  |  Particle momentum
         PMOFLK (NPFLKA) = PBEAM
*        PMOFLK (NPFLKA) = SQRT ( TKEFLK (NPFLKA) * ( TKEFLK (NPFLKA)
*    &                           + TWOTWO * AM (IONID) ) )
*  |  +-------------------------------------------------------------*
*  |  |  Check if it is a neutrino, if so force the interaction
*  |  |  (unless the relevant flag has been disabled)
         IF ( LISNUT (IJBEAM) .AND. LNUFIN ) THEN
            LFRPHN (NPFLKA) = .TRUE.
*  |  |
*  |  +-------------------------------------------------------------*
*  |  |  Not a neutrino
         ELSE
            LFRPHN (NPFLKA) = .FALSE.
         END IF
*  |  |
*  |  +-------------------------------------------------------------*
      END IF
*  |
*  +-------------------------------------------------------------------*
*  From this point .....
*  Particle generation (1 for primaries)
      LOFLK  (NPFLKA) = 1
*  User dependent flag:
      LOUSE  (NPFLKA) = 0
*  No channeling:
      KCHFLK (NPFLKA) = 0
      ECRFLK (NPFLKA) = ZERZER
*  Extra infos:
      INFSTK (NPFLKA) = 0
      LNFSTK (NPFLKA) = 0
      ANFSTK (NPFLKA) = ZERZER
*  Parent variables:
      IPRSTK (NPFLKA) = 0
      EKPSTK (NPFLKA) = ZERZER
*  User dependent spare variables:
      DO 100 ISPR = 1, MKBMX1
         SPAREK (ISPR,NPFLKA) = ZERZER
 100  CONTINUE
*  User dependent spare flags:
      DO 200 ISPR = 1, MKBMX2
         ISPARK (ISPR,NPFLKA) = 0
 200  CONTINUE
*  Save the track number of the stack particle:
      ISPARK (MKBMX2,NPFLKA) = NPFLKA
      NPARMA = NPARMA + 1
      NUMPAR (NPFLKA) = NPARMA
      NEVENT (NPFLKA) = 0
      DFNEAR (NPFLKA) = +ZERZER
*  ... to this point: don't change anything
      AKNSHR (NPFLKA) = -TWOTWO
*  Cosines (tx,ty,tz)
      TXFLK  (NPFLKA) = UBEAM
      TYFLK  (NPFLKA) = VBEAM
      TZFLK  (NPFLKA) = WBEAM
*     TZFLK  (NPFLKA) = SQRT ( ONEONE - TXFLK (NPFLKA)**2
*    &                        - TYFLK (NPFLKA)**2 )
*  Polarization cosines:
      TXPOL  (NPFLKA) = -TWOTWO
```

```
      TYPOL  (NPFLKA) = +ZERZER
      TZPOL  (NPFLKA) = +ZERZER
*  Particle coordinates
      XFLK   (NPFLKA) = XBEAM
      YFLK   (NPFLKA) = YBEAM
      ZFLK   (NPFLKA) = ZBEAM
*  Calculate the total kinetic energy of the primaries: don't change
*  +-------------------------------------------------------------------*
*  |  (Radioactive) isotope:
      IF ( IJBEAM .EQ. -2 .AND. LRDBEA ) THEN
*  |
*  +-------------------------------------------------------------------*
*  |  Heavy ion:
      ELSE IF ( ILOFLK (NPFLKA) .EQ. -2 .OR.
     &          ILOFLK (NPFLKA) .GT. 100000 ) THEN
         TKESUM = TKESUM + TKEFLK (NPFLKA) * WTFLK (NPFLKA)
*  |
*  +-------------------------------------------------------------------*
*  |  Standard particle:
      ELSE IF ( ILOFLK (NPFLKA) .NE. 0 ) THEN
         TKESUM = TKESUM + ( TKEFLK (NPFLKA) + AMDISC (ILOFLK(NPFLKA)) )
     &          * WTFLK (NPFLKA)
*  |
*  +-------------------------------------------------------------------*
*  |
      ELSE
         TKESUM = TKESUM + TKEFLK (NPFLKA) * WTFLK (NPFLKA)
      END IF
*  |
*  +-------------------------------------------------------------------*
      RADDLY (NPFLKA) = ZERZER
*  Here we ask for the region number of the hitting point.
*     NREG (NPFLKA) = ...
*  The following line makes the starting region search much more
*  robust if particles are starting very close to a boundary:
      CALL GEOCRS ( TXFLK (NPFLKA), TYFLK (NPFLKA), TZFLK (NPFLKA) )
      CALL GEOREG ( XFLK  (NPFLKA), YFLK  (NPFLKA), ZFLK  (NPFLKA),
     &              NRGFLK(NPFLKA), IDISC )
*  Do not change these cards:
      CALL GEOHSM ( NHSPNT (NPFLKA), 1, -11, MLATTC )
      NLATTC (NPFLKA) = MLATTC
      CMPATH (NPFLKA) = ZERZER
      CALL SOEVSV
      RETURN
*=== End of subroutine Source =========================================*
      END
```

The “new” source routine

- Distributed since FLUKA4-1.0 release
- Simplified appearance
- Long & meaningful names for variables and routines
- Use of **`implicit none`** (see later)
- Abundant comments and examples
- Advanced sampling routines

Removed from snapshot

- Variables for user’s usage clearly indicated
- Lines not to be edited are “hidden” in routines

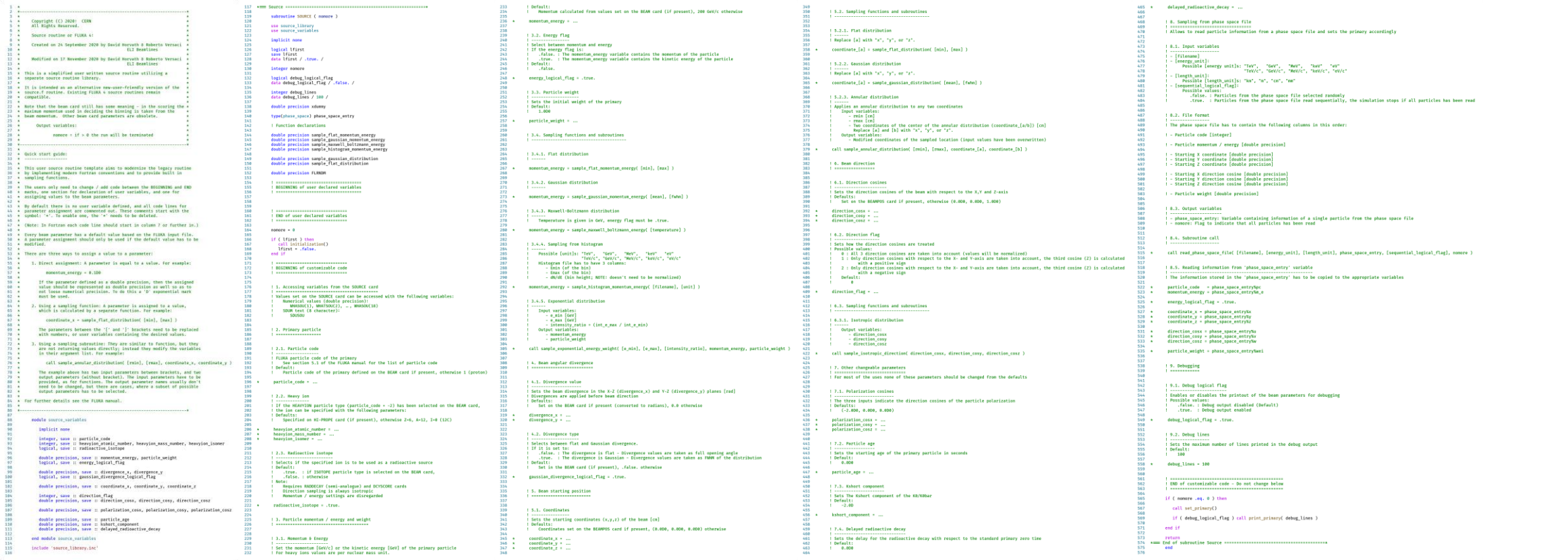
  in the **`source_library.inc`** library file

- **Old source routines can still be used**

```
      ! ============================
      ! BEGINNING of customizable code
      ! ============================

*     particle_code = ...

*     heavyion_atomic_number = ...
*     heavyion_mass_number = ...
*     heavyion_isomer = ...

*     radioactive_isotope = .true.

*     momentum_energy = ...

*     energy_logical_flag = .true.

*     particle_weight = ...

*     divergence_x = ...
*     divergence_y = ...

*     gaussian_divergence_logical_flag = .true.

*     coordinate_x = ...
*     coordinate_y = ...
*     coordinate_z = ...

*     direction_cosx = ...
*     direction_cosy = ...
*     direction_cosz = ...

*     direction_flag = ...

*     polarization_cosx = ...
*     polarization_cosy = ...
*     polarization_cosz = ...

*     particle_age = ...
*     kshort_component = ...
*     delayed_radioactive_decay = ...

      ! ============================================
      ! END of customizable code - Do not change below
      ! ============================================

      if ( nomore .eq. 0 ) then

         call set_primary()

         if ( debug_logical_flag ) call print_primary( debug_lines )

      end if

      return
*=== End of subroutine Source ========================================*
      end
```

The “new” source routine

- Without removing comments, examples and advanced features (notice the ratio of code and comment lines)
- Note: the snapshot is not meant to be read – Detailed view will follow



History of Fortran

- Fortran born in the early 1950s, and the first compiler was released in 1957

Standards:

- Fortran 66 – The first standard
- Fortran 77 – Extension on Fortran 66
- Fortran 90 – Dynamic memory allocation / introduction of the *Free* format
- Fortran 95 – High performance Fortran specification
- Fortran 2003 – Object oriented programming
- Fortran 2008 / 2018 – Extensions of Fortran 2003

FLUKA is still mostly (if not fully) compatible Fortran 77
This doesn’t mean that we can’t use newer things in our user routines

(Unexpected) Features and limitations of Fortran (77)

- Source file format
  - Fixed
  - Free
- Naming convention
- Subprograms
  - Functions
  - Subroutines
- Variable declaration
  - Implicit
  - Explicit

Source file format

- Fortran 77 uses the *Fixed* file format (extensions: **.f** or **.for**):
  - Maximum 78 characters in one line
  - First 6 are reserved for special function:
    - If the first character is ‘c’ or ‘*’, then the line is a comment
    - If the 6th position is not empty, then the line is treated as a continuation of the previous one (Often the ‘&’ character is used)
  - With the gfortran compiler it is possible to increase the maximum line length
    - In FLUKA 4 it is extended to 132 characters

- Fortran 90 introduced the *Free* format (extensions: **.f90**, [**.f95**, etc.]):
  - Code can start at the 1st position

- *Note*: It is not possible to mix both in the same source file.
  Gfortran compiler expects the “correct” format based on the file extension.

Naming convention

- Fortran 77 variable and (subprogram) names:
  - Limited to 6 alphanumerical characters
  - Have to start with a letter
  - Case insensitive

- Starting with Fortran 90 the variable names
  - Can be up to 31 character long
  - Can contain letters, numbers and underscore ('_')
  - Have to start with a letter
  - Case insensitive

Feature exploited in the new source routine

- *Note*: Try to use descriptive names, to make code readable

Subprograms

- Two types:
  - Function
    - Has a return value
    - Used in assignment: `variable = function(input_variable_1, …)`
  - Subroutine
    - Doesn't have a return value
    - Accessible with the CALL statement: `call subroutine(input_variable_1, …)`

- Passing variables
  - In Fortran you pass the variable, not the value of the variable (Like passing a pointer in C)
  - This means the subprograms may irreversibly modify the value of the input variables
    - Desired behavior if you want to return multiple variables
    - Can lead to side effects

Variable declaration

- Fortran by default uses *implicit declaration*, which means the type of the variable (integer, real, etc.) is determined by a preset rule.
- The default rule is:
  - If the variable starts with the letter I, J, K, L, M, or N it is an integer
  - Otherwise, it is a real (single precision float)
- In FLUKA however:
  - Variables with the 1st letter I, J, K, L, M, and N are still integers
  - But the others are double precision (floats)

- It is possible (and necessary) to overwrite this with *explicit declaration*, where you manually specify the type of the variable, like:

```
double precision my_intensity
logical my_flag
```

Variable declaration

- Biggest issue is that typos remain hidden:
  - If you have a typo in a variable name, the compiler won't raise an error
  - It is a different, but valid variable without a value
  - Using it in calculations will lead to unexpected results
- Other issue is the unexpected type conversion:
  - For example: Information is lost if you want to assign a double precision number to INTEGER
- Solution in the "new" source routine: `implicit none`
  - This statement disables the implicit declaration, and every variable has to me manually declared
  - Exception: FLUKAs built in variables don't need to be declared in the source routine (they will remain implicitly declared)
- Convention in the "new" source routine:
  - Variables with uppercase names: FLUKA variables
  - Variables with lowercase names: explicitly declared variables

Numbers and Constants in User routines

- To keep the high accuracy of the calculation
  - Every variable containing a floating-point number should have the type *double precision*
  - The assigned numbers should also be double precision:

    For example: **`radius = 2.0D0`**, or in a function: **`variable = function(1.0D0)`**

    The ‘D’ character indicated, that this is number should be treated as double precision.

    If it is ‘E’ or missing, then the number will be single precision

- To simplify writing numbers FLUKA already defined many numbers as variables:
  - **`ONEONE = 1.0D0`**
  - **`TWOTWO = 2.0D0`**
  - **`HLFHLF = 0.5D0`**
  - **`PIPIPI = `** π **` = 3.141592`**…
  - **`TWOPIP = `** 2π **` = 6.283185`**…

  Full list available in the **`dblprc.inc`** include file

Source routine – Initialization

```
154   ! ====================================
155   ! BEGINNING of user declared variables
156   ! ====================================
157
158
159
160   ! ==================================
161   ! END of user declared variables
162   ! ==================================
```

- Dedicated space for the declaration of user variables (and functions)

Source routine – Initialization

```
166   if ( lfirst ) then
167      call initialization()
168      lfirst = .false.
169   end if
```

- Initialization of internal variables
- Only performed the first time the routine is called

- To overwrite the default values the relevant lines needs to be uncommented, by removing the '*' at the beginning of the line.
  (See next slides)

Source routine – Primary particle

```
196   *      particle_code =  ...
```

- By default, the particle type given in the **BEAM** card is taken
- Particle codes explained in FLUKA manual section 5.1
- Possible application: beam made of more than one type particles

```
206   *      heavyion_atomic_number =  ...
207   *      heavyion_mass_number =  ...
208   *      heavyion_isomer =  ...
```

- Only used if primary particle is set to HEAVYION or ISOTOPE
- Default values are set on the **HI-PROPE** card, or for ^{12}C if the card is missing

Source routine – Energy / momentum

```
236   *      momentum_energy = ...
```

- By default, the particle momentum is expected
- The default value is based on the **BEAM** card
  (Automatically converted into momentum if energy is given on the **BEAM** card)

- If energy is specified in the source routine, the following logical value must be set *.true.*

```
248   *      energy_logical_flag = .true.
```

Source routine – Energy / momentum

- The momentum divergence set on the **BEAM** card is not retained
- It in necessary to specify in the source routine
- It is easy with the supplied functions / subroutine

Flat spectrum:

```
267   *      momentum_energy = sample_flat_momentum_energy( [min], [max]
```

Gaussian spectrum:

```
273   *      momentum_energy = sample_gaussian_momentum_energy( [mean],
```

Maxwell-Boltzmann spectrum:

```
280   *      momentum_energy = sample_maxwell_boltzmann_energy( [tempera
```

Spectrum from histogram:

```
292   *      momentum_energy = sample_histogram_momentum_energy( [filena
```

Exponential spectrum:
(biased sampling)

```
305   *      call sample_exponential_energy_weight( [e_min], [e_max], [i
```

Source routine – Particle weight

```
257   *      particle_weight = ...
```

- Monte Carlo concept for biased sources

- The default value (**`particle_weight = 1.0`**) is usually sufficient
- Not for a beginners' use, mentioned here for completeness

- Note: The exponential spectrum sampling subroutine, uses variable particle weight

Source routine – Beam divergence

```
319   *      divergence_x =  ...
320   *      divergence_y =  ...
```

- By default:
  - values are taken from the **BEAM** card
  - It is assumed to be a flat angular distribution

- For Gaussian divergence the following logical value must be set *.true.*

```
332   *      gaussian_divergence_logical_flag = .true.
```

Source routine – Beam starting position

```
345   *       coordinate_x =  ...
346   *       coordinate_y =  ...
347   *       coordinate_z =  ...
```

- By default, values are taken from the **BEAMPOS** card

- Beam shape set on the **BEAM** card, and
- Extended sources specified on additional **BEAMPOS** cards are not implemented

Source routine – Beam starting position

- Some predefined routines (2 functions and 1 subroutine) are already available:

Flat distribution:

```
358   *      coordinate_[a] = sample_flat_distribution( [min], [max] )
```

Gaussian distribution:

```
365   *      coordinate_[a] = sample_gaussian_distribution( [mean], [fwhm] )
```

Annular distribution:

```
379   *      call sample_annular_distribution( [rmin], [rmax], coordinate_[a],
```

Remember the values must be in double precision (`1.0D0`).

Note: If annular sampling is used, the coordinates has to be set manually as well.

Source routine – Beam direction

```
392   *      direction_cosx = ...
393   *      direction_cosy = ...
394   *      direction_cosz = ...
```

- By default, values are taken from the **BEAMPOS** card

- If the **direction_flag** is set to:

```
409   *      direction_flag = ...
```

  - 0 : All three values are considered and the they are normalized automatically (Default)
  - 1 : The manually set value of the z direction is disregarded. Instead, it is calculated from the x and y direction cosines with a positive sign.
  - 2 : As with option 1, but negative sign is used.

- A predefined subroutine is are already available for isotropic direction sampling

```
422   *      call sample_isotropic_direction( direction_cosx, direction_cosy, direction_cosz )
```

Source routine – Debugging

- To help debug the source routine, the major particle parameters can be printed
- To enable this feature, set

```
549   *      debug_logical_flag = .true.
```

- The printed parameters:
  - Energy / momentum
  - Coordinates
  - Direction
  - Weight

- The number of primaries printed can be set with:

```
558   *      debug_lines = 100
```

Some predefined FLUKA random sampling routines

- FLUKA offers some useful, predefined routines for random sampling

- `my_variable = FLRNDM(XDUMMY)`

  Assigns a 64-bit random number in [0,1)

- `call FLNRRN(gauss1)`

  Returns a Gaussian distributed random number

- `call FLNRR2(gauss1,gauss2)`

  Returns two uncorrelated Gaussian distributed random numbers

- `call SFECFE(sint,cost)`

  Returns sine and cosine of a random azimuthal angle

SOURCE card and passing parameters

- To invoke a source routine, it is necessary to add a **SOURCE** card
- A **SOURCE** card can be empty or can be used to pass parameters to the routine
- Max. 18 numerical values (**WHASOU(ii)**) and 1 string (max. 8 characters) (**SDUSOU**) can be

SOURCE	#1: 7.	#2: 250.	#3: 12.5
sdum: linksour	#4: 3.75	#5:	#6:
	#7:	#8:	#9:
	#10:	#11:	#12:
	#13:	#14:	#15:
	#16:	#17:	#18:

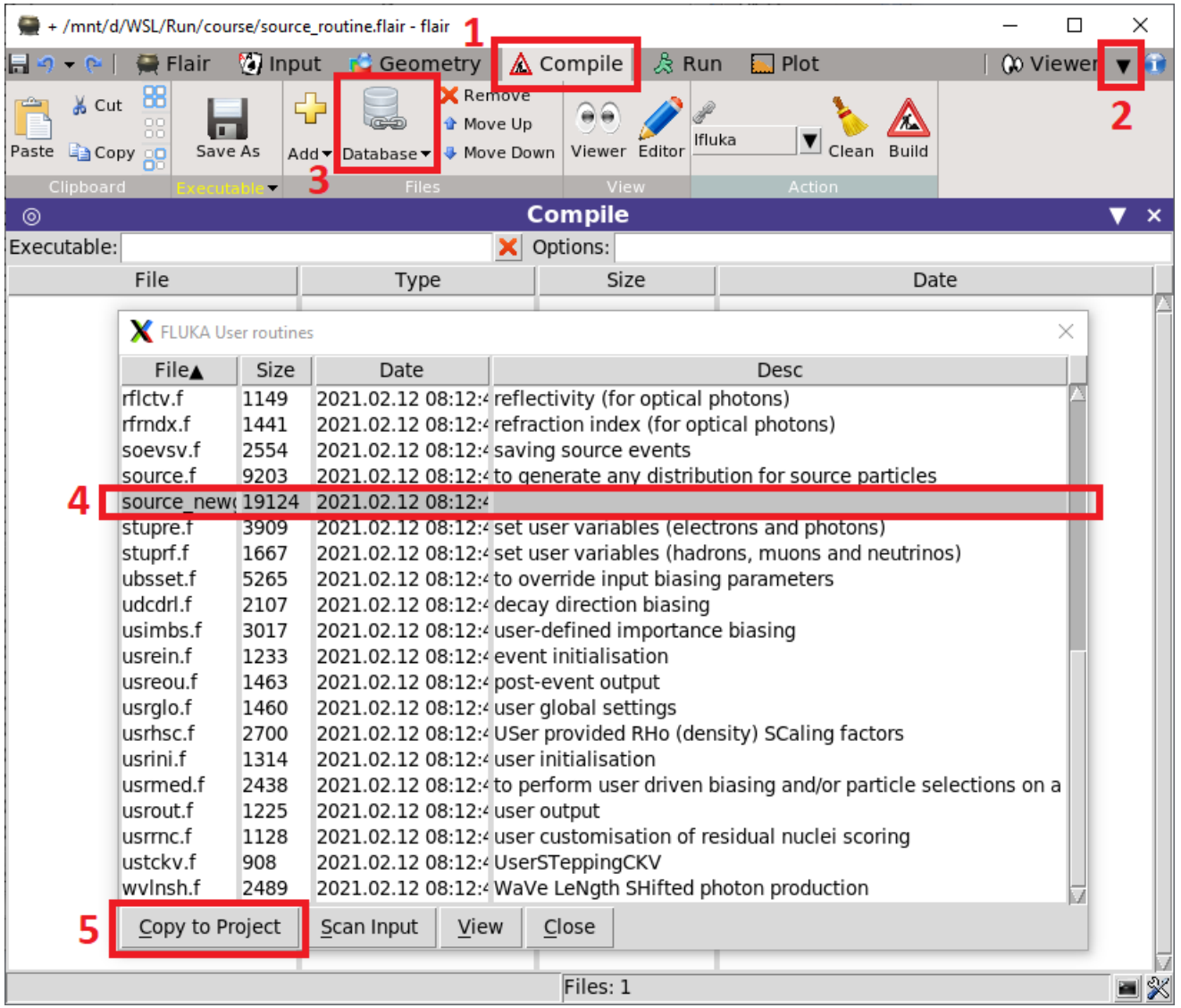
- Good practice advice:

  Even if the beam energy / momentum is defined in the source routine, specify it in the **BEAM** card as it is used for internal initialization.
  Set a momentum value higher than the maximum possible one.

Adding the user routine to the project folder

1. Open [Compile] tab
2. It is maybe hidden in the dropdown menu
3. Click the [Database] button (Use [Add] for an existing file)
4. Select the user routine you want to use
5. Click [Copy to Project]

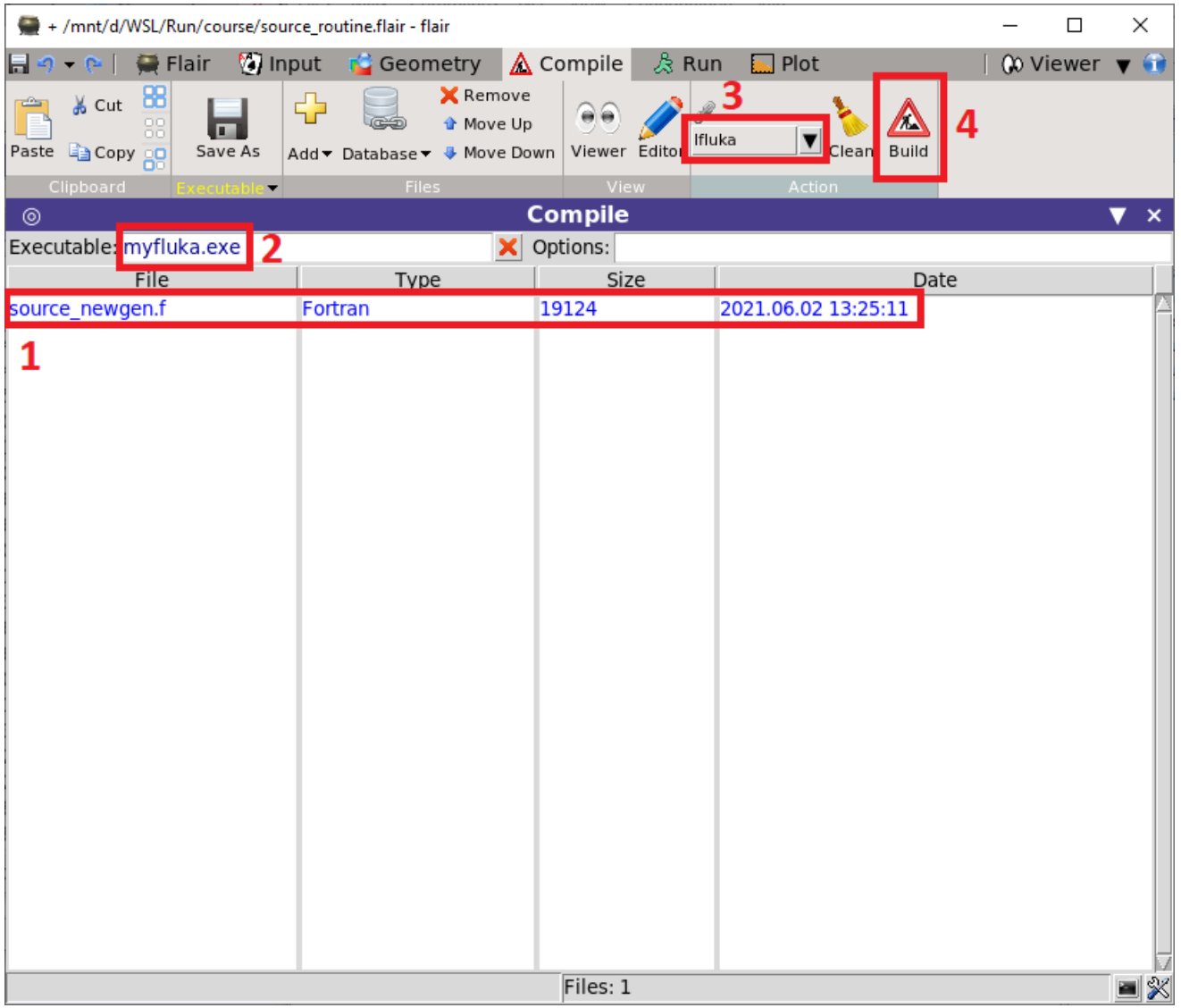
The copied user routine will be in the Flair projects directory



Compiling a custom FLUKA executable

1. Verify that the user routine is in the list
2. Name your custom executable
3. Select the appropriate linker:
   a. Use *lfluka* by default
   b. Use *ldpmqmd* if DPMJET or RQMD models are needed
4. Compile the executable

The custom executable should be set default on the [Run] tab automatically



Time to do some hands-on practice!

- We will now see together a few small examples of “new” source routine

xkcd.com/303


FLUKA