Source routine

An introduction to a new approach to source routines

Why do we need a source routine

- The source routine is used to define complex sources, when the options provided in the **BEAM**, **BEAMPOS** and **BEAMAXES** cards are not enough.

- Most common use cases:
  - Mixed field
  - Beam with an energy spectrum
  - Complex beam shape
  - Second step of a two-step simulation

The “old” source routine

- Scary for beginners, limited documentation
- Written according the FORTRAN 77 standard, prone to errors

```
*
*=== 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
```

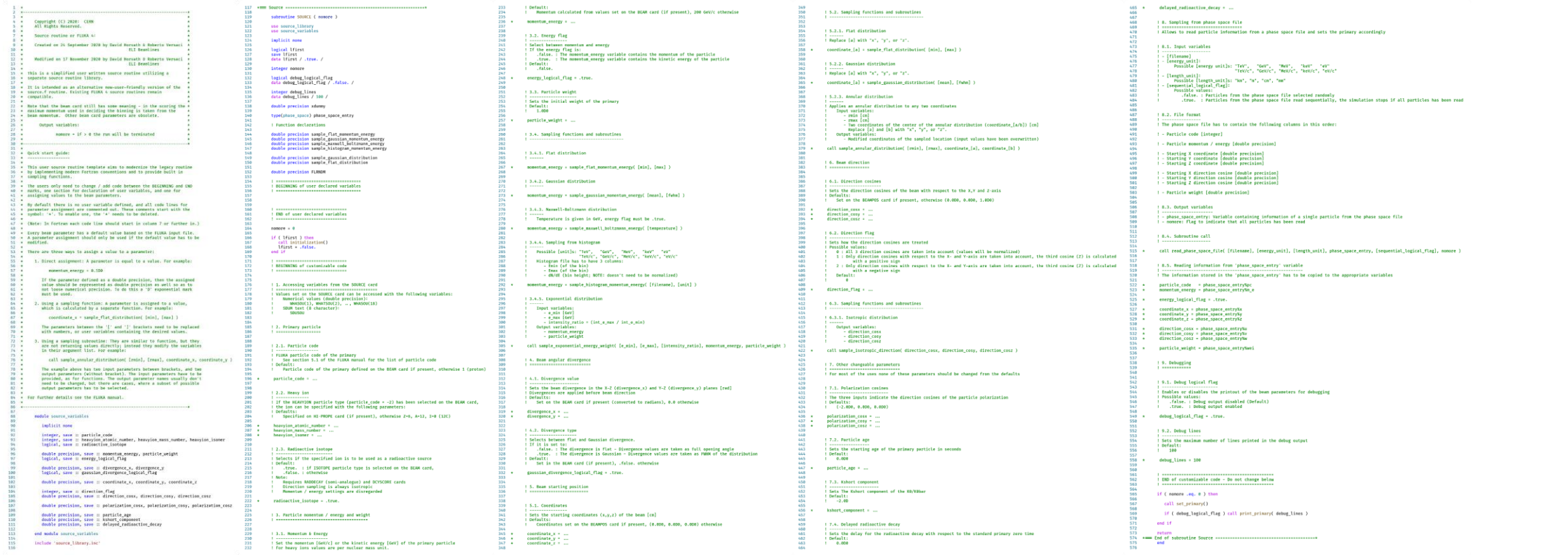
The “new” source routine

- Simplified appearance
- Long & meaningful names for variables and routines
- Forced declaration of variables – Use of **`implicit none`**
- Documented by comments and in the manual
- 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**

The “new” source routine



- Note: the snapshot is not meant to be read – Detailed view will follow

User declaration

```
!  ===================================
!  BEGINNING of user declared variables
!  ===================================
```

```
!  =================================
!  END of user declared variables
!  =================================
```

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

User declaration (example)

```
! ====================================
! BEGINNING of user declared variables
! ====================================

integer :: counter
double precision :: energy
logical :: flag

! ==================================
! END of user declared variables
! ==================================
```

Initialization

- Initialization of internal variables
  Runs every time, resetting their values to the defaults
- Custom initialization block

  Runs only the first time the source routine is used

```
call initialization()

if ( first_run ) then

   ! ═══════════════════════════════════════
   ! BEGINNING of custom initialization
   ! ═══════════════════════════════════════



   ! ═══════════════════════════════════
   ! END of custom initialization
   ! ═══════════════════════════════════

   first_run = .false.
end if
```

Main section

- For setting the internal variables directly, or using one of the sampling routines
- To enable a line, remove the `*` (The command should start on the 7th column)
- The variables with `[` `]` brackets and … are placeholders, they need to be replaced with values or user variables (Brackets should be deleted)
- Always use double precision format for floating point numbers (1.0d0)

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

*     particle_code = ...

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

*     momentum_energy = ...

*     energy_logical_flag = .true.

*     particle_weight = ...

*     momentum_energy = sample_flat_momentum_energy( [min], [max] )
*     momentum_energy = sample_gaussian_momentum_energy( [mean], [fwhm] )
*     momentum_energy = sample_maxwell_boltzmann_energy( [temperature] )
*     momentum_energy = sample_histogram_momentum_energy( [filename], [unit] )
*     momentum_energy = sample_spectrum_momentum_energy( [filename], [unit] )

*     call sample_exponential_energy_weight( [e_min], [e_max], [intensity_ratio], momentum_energy, particle_weight )

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

*     gaussian_divergence_logical_flag = .true.

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

*     coordinate_[a] = sample_flat_distribution( [min], [max] )
*     coordinate_[a] = sample_gaussian_distribution( [mean], [fwhm] )
*     call sample_annular_distribution( [rmin], [rmax], coordinate_[a], coordinate_[b] )

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

*     direction_flag = ...

*     call sample_isotropic_direction( direction_cosx, direction_cosy, direction_cosz )

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

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

*     call read_phase_space_file( [filename], [energy_unit], [length_unit], phase_space_entry, [sequential_logical_flag], nomore )

*     particle_code   = phase_space_entry%pc
*     momentum_energy = phase_space_entry%m_e

*     energy_logical_flag = .true.

*     coordinate_x = phase_space_entry%x
*     coordinate_y = phase_space_entry%y
*     coordinate_z = phase_space_entry%z

*     direction_cosx = phase_space_entry%u
*     direction_cosy = phase_space_entry%v
*     direction_cosz = phase_space_entry%w

*     particle_weight = phase_space_entry%wei

*     debug_logical_flag = .true.
*     debug_lines = ...
```

Primary particle

```
*      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 of particles

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

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

Energy / momentum

```
*      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 in the **BEAM** card)

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

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

Energy / momentum

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

Sampling functions:

Analytical

- Flat (Uniform)
- Gaussian
- Maxwell-Boltzmann
- Exponential

From an external file

- Histogram
- Continuous spectrum
- Discrete spectrum

Energy / Momentum – Analytical samplings

- Flat / uniform:
  - Samples uniformly between two momentum [GeV/c] or, energy [GeV] values

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

- Gaussian:
  - Samples from a Gaussian distribution with a given mean and FWHM value ([GeV/c] or [GeV])

```
*       momentum_energy = sample_gaussian_momentum_energy( [mean], [fwhm] )
```

- Maxwell-Boltzmann:
  - Samples from a Maxwell-Boltzmann energy distribution with a given temperature [GeV]
  - It is only meaningful if the **`energy_logical_flag`** is set to `.true.`

```
*       momentum_energy = sample_maxwell_boltzmann_energy( [temperature] )
```

Energy / Momentum – Analytical samplings

- Exponential:
  - Samples according the exponential distribution between two energies [GeV], with a given intensity ration at the specified energies

```
*      call sample_exponential_energy_weight(
                [e_min], [e_max], [intensity_ratio],
                momentum_energy, particle_weight
             )
```

  - It is only meaningful if the **`energy_logical_flag`** is set to **`.true.`**
  - Note 1: Different syntax used (function vs. subroutine)
  - Note 2: The return values **`momentum_energy`** and **`particle_weight`** are among the arguments
  - Note 3: This is a biased sampling! It is not suitable for cases where fully analogue simulation is required (E.g. scoring with **`DETECT`** card)

Energy / Momentum – Sampling from file

- Histogram:
  - Samples from a histrogram specified in an external file

```
*       momentum_energy = sample_histogram_momentum_energy(
           [filename], [unit]
        )
```

  - The external file needs to have 3 columns:
    - Lower energy boundary of the histogram's bins
    - Upper energy boundary of the histogram's bins
    - Intensity per energy unit (dN/dp or dN/dE)
  - The particle momentum / energy is sampled uniformly withing a bin
  - `[unit]` variable is the unit used in the external file. Possible values:
    - TeV/c, GeV/c, MeV/c, keV/c, J
    - TeV, GeV, MeV, keV, eV, J

Energy / Momentum – Sampling from file

- Continuous spectrum:
  - Samples from a continuous spectrum specified in an external file

```
*       momentum_energy = sample_spectrum_momentum_energy(
           [filename], [unit]
        )
```

  - Samples from a discrete (line) spectrum specified in an external file

```
*       momentum_energy = sample_discrete_momentum_energy(
           [filename], [unit]
        )
```

  - The external file needs to have 2 columns:
    - Energy
    - Intensity at the specified energy
  - With the continuous spectrum, the intensity is linearly interpolated

Energy / Momentum – Examples

- Setting energy flag to true:

```
energy_logical_flag = .true.
```

- Monoenergetic beam:

```
momentum_energy = 1.0d-1
```

- Gaussian beam:

```
momentum_energy = sample_gaussian_momentum_energy(1.0d-1, 1.0d-2)
```

Energy / Momentum – Examples

- Sampling from an exponential distribution:

```
      call sample_exponential_energy_weight(1.0d-6, 1.0d-3, 0.01d0,
     &   momentum_energy, particle_weight)
```

- Sampling from an external spectrum:

```
      momentum_energy = sample_spectrum_momentum_energy(
     &   'spectrum.txt', 'MeV')
```

Note: The `&` character is for the line continuations, it should be always in column 6.

Source routine – Particle weight

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

- To create biased sources (see Monte Carlo lecture)
- Usually needed only for complex source definitions
- Note: The exponential spectrum sampling subroutine uses variable particle weight, but in that case, it is automatically set

Source routine – Beam divergence

```
*      divergence_x =  ...
*      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.*

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

Source routine – Beam starting position

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

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

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

Source routine – Beam starting position

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

Flat distribution:

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

Gaussian distribution:

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

Annular distribution:

```
*      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 have to be selected as well.

Source routine – Beam direction

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

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

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

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

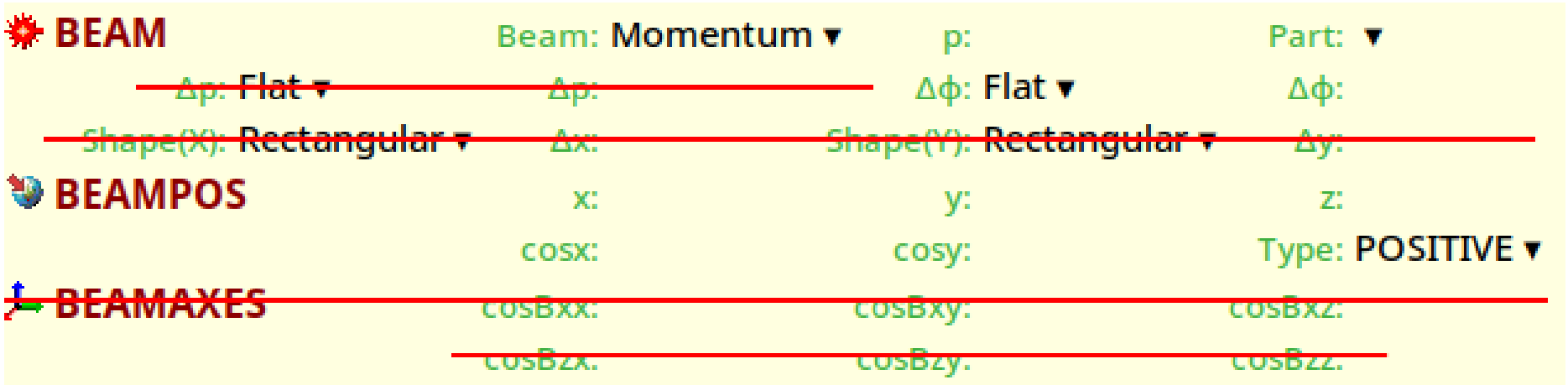
  - 0 : All three values are considered and the they are normalized automatically (Default)
  - 1 : The manually set value of the z direction is disregarded. Instead, it is calculated from the x and y direction cosines with a positive sign.
  - 2 : As with option 1, but negative sign is used.

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

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

Source routine – Unused values

- It is important to remember, not all values used in the FLUKA input are used in the source routine:
  - The beam momentum distribution
  - The shape of the extended beam / volumetric sources
  - The separate coordinate system set up for the beam



- If one of these features is required, it needs to be programmed in the source routine as well by using the available sampling procedures or by custom code.

Source routine – Phase-space sampling

- Used for the second step in a two-step simulation

- It reads a file containing information on individual particles:
  - Particle code
  - Momentum / energy
  - Starting coordinate
  - Starting direction
  - Weight

- Can replay the particles sequentially, or select from them randomly

```
*      call read_phase_space_file( [filename], [en

*      particle_code   = phase_space_entry%pc
*      momentum_energy = phase_space_entry%m_e

*      energy_logical_flag = .true.

*      coordinate_x = phase_space_entry%x
*      coordinate_y = phase_space_entry%y
*      coordinate_z = phase_space_entry%z

*      direction_cosx = phase_space_entry%u
*      direction_cosy = phase_space_entry%v
*      direction_cosz = phase_space_entry%w

*      particle_weight = phase_space_entry%wei
```

Source routine – Debugging

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

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

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

- The number of primaries printed can be set with:

```
*      debug_lines = 100
```

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**)

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:

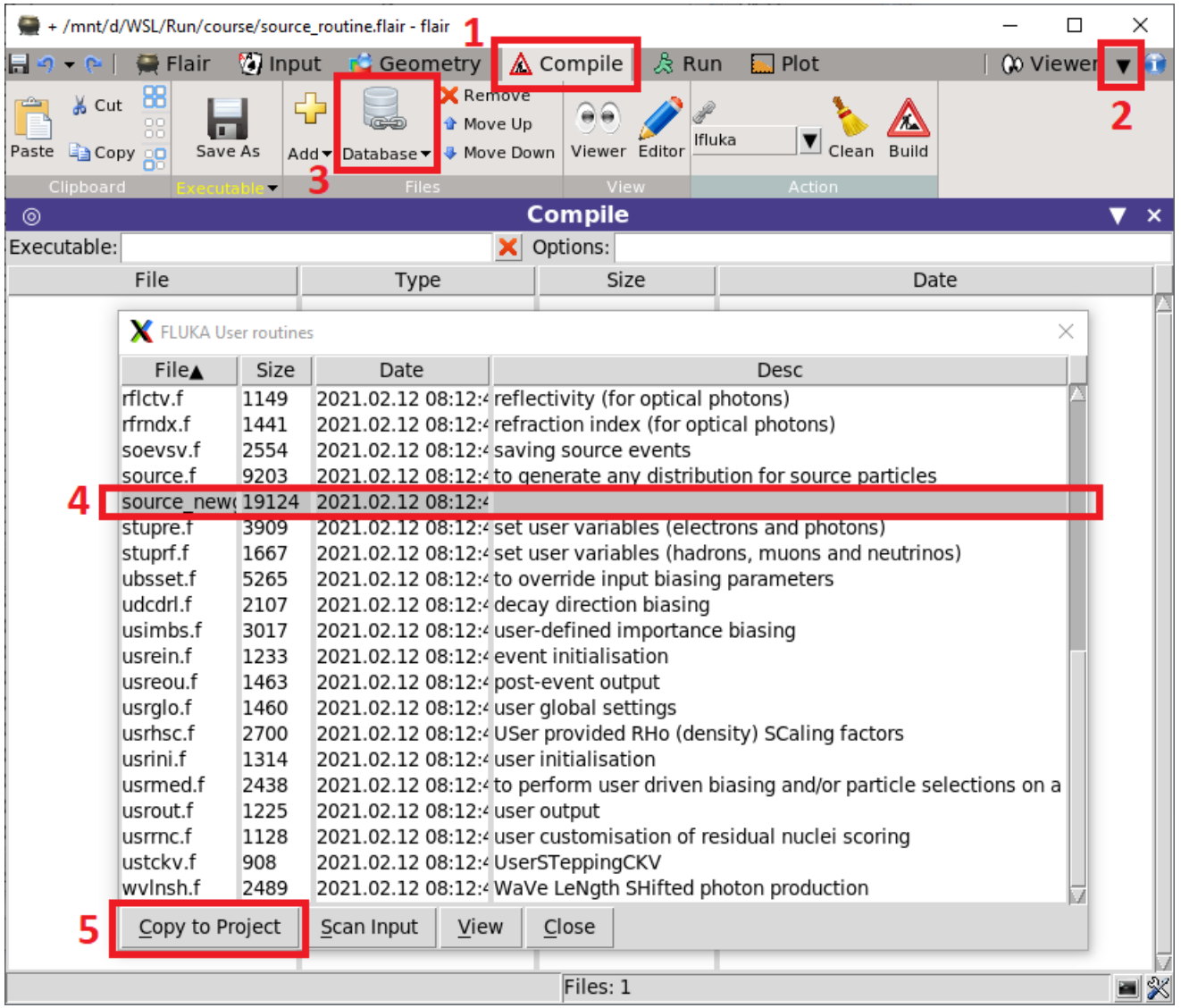
- Good practice:

  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]

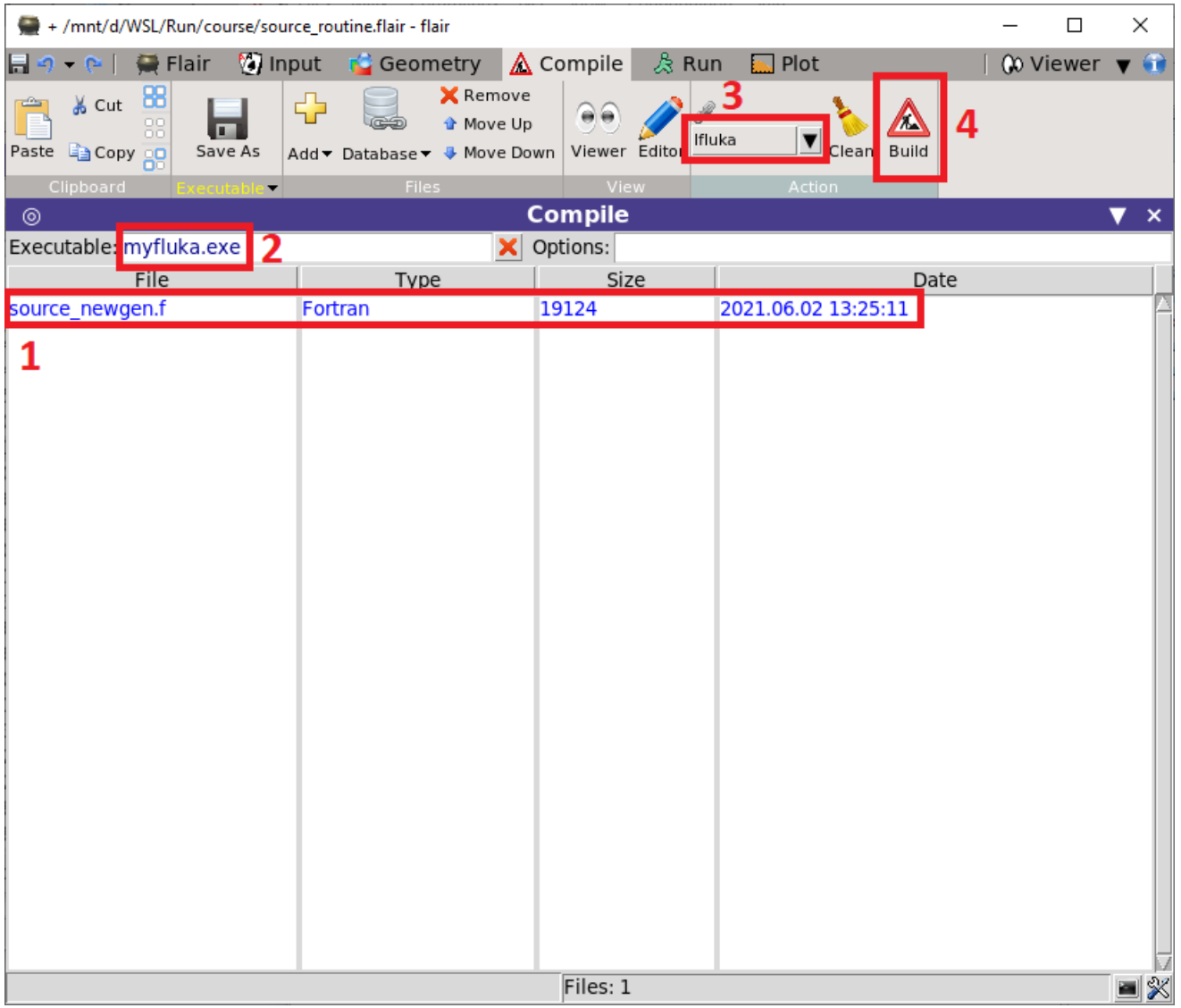
The copied user routine will be in the Flair projects directory



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 for an exercise!

xkcd.com/303


FLUKA