



The source routine

A refresher from the beginners' course, plus something more

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 `source_newgen` routine

- Starting from FLUKA4-1.0 release, a “new” source routine is distributed
- Simplified appearance
- Long & meaningful names for variables and routines
- Use of `implicit none` (see later)
- Well 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**

User declaration & variable initialization

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

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

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

```
call initialization()  
  
if ( first_run ) then  
  
! =====  
! BEGINNING of custom initialization  
! =====  
  
! =====  
! END of custom initialization  
! =====  
  
first_run = .false.  
end if
```

- Initialization of **internal** variables
Runs for every event & resets default
- Custom initialization block
Runs only at the first call

User declaration & variable initialization

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

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

example

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

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

```
call initialization()
```

```
if ( first_run ) then
```

```
! =====  
! BEGINNING of custom initialization  
! =====
```

```
! =====  
! END of custom initialization  
! =====
```

```
first_run = .false.
```

```
end if
```

- Initialization of **internal** variables
Runs for every event & resets default
- Custom initialization block
Runs only at the first call

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%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 are 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 to **.true.**

```
energy_logical_flag = ...
```

Energy / momentum

- The momentum divergence set on the **BEAM** card is not retained
- It is 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 ratio 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 returned 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)

Interjection

The math behind sampling from most of the functions

Sampling from an arbitrary function

- Have the integrable function $f(x)$ as a probability density function
- Calculate the cumulative distribution function (CDF):

$$F(x) = \frac{\int_{x_{min}}^x f(t)dt}{\int_{x_{min}}^{x_{max}} f(t)dt}$$

- Sample a random number (ξ) uniformly between 0 and 1, making $F(x) = \xi$
- Invert the CDF to get $x = F^{-1}(\xi)$

Sampling from an arbitrary function - Example

- Take the following function where $\lambda > 0$:

$$f(x) = \begin{cases} 0, & \text{if } x < 0 \\ e^{-x/\lambda}, & \text{if } 0 \leq x \end{cases}$$

- The indefinite integral:

$$\int e^{-t/\lambda} dt = -\lambda e^{-t/\lambda} + c$$

- Definite integrals:

$$\int_0^x e^{-t/\lambda} dt = \left[-\lambda e^{-t/\lambda} \right]_0^x = \lambda \left(1 - e^{-x/\lambda} \right)$$

$$\int_0^{\infty} e^{-t/\lambda} dt = \left[-\lambda e^{-x/\lambda} \right]_0^{\infty} = \lambda$$

Sampling from an arbitrary function - Example

- Cumulative distribution function:

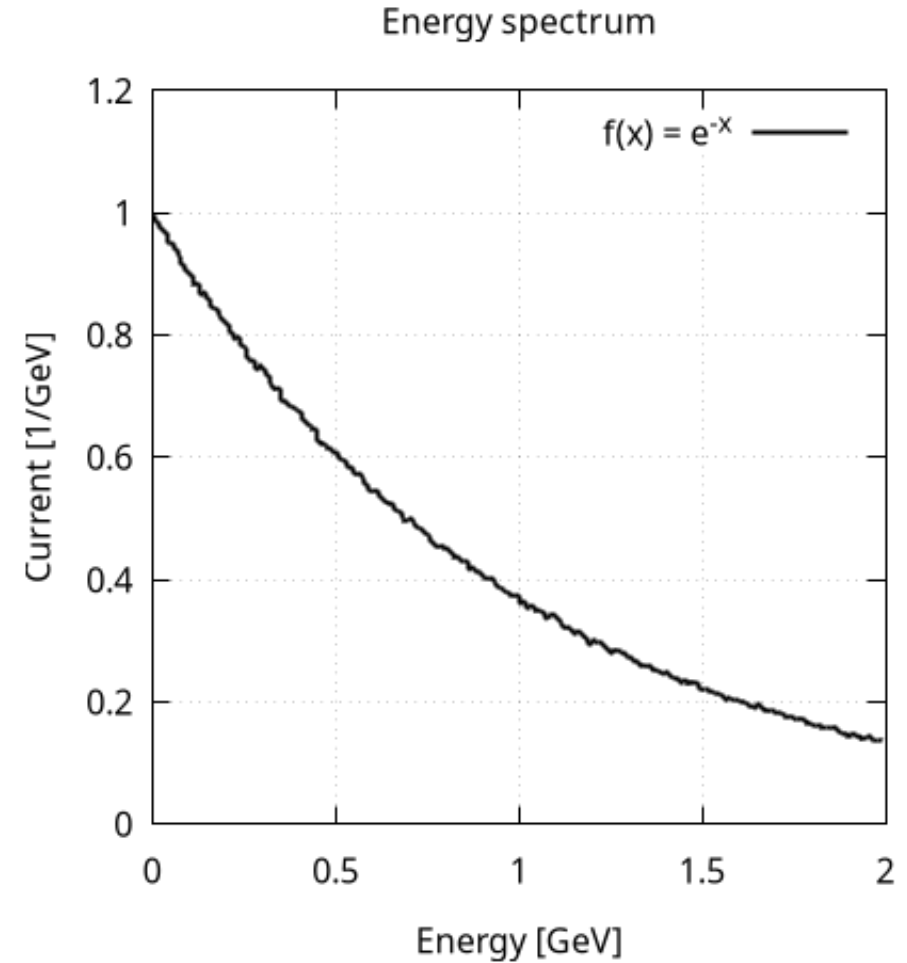
$$F(x) = 1 - e^{-x/\lambda}$$

- Sampling a uniformly distributed random number between 0 and 1:

$$1 - e^{-x/\lambda} = \xi$$

- Inverse function:

$$x = -\lambda \ln(1 - \xi)$$



End of interjection

Energy / Momentum – Sampling from file

- Histogram:

- Samples from a histogram specified in an external file

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

- The external file should 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 within a bin
- **[unit]** variable is the momentum / energy 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 should 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 line continuation, it should always be in column 6.

Particle weight

```
particle_weight = ...
```

- To create biased sources (see Monte Carlo and Biasing lectures)
- 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

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.
```

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 are not implemented
- Extended sources specified in additional **BEAMPOS** cards are not implemented

Beam starting position

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

Flat distribution:

```
coordinate_[a] = sample_flat_distribuiton( [min], [max] )
```

Gaussian distribution:

```
coordinate_[a] = sample_gaussian_distribuiton( [mean], [fwhm] )
```

Annular distribution:

```
call sample_annular_distribuiton( [rmin], [rmax],  
                                coordinate_[a], coordinate_[b] )
```

Remember the values must be in double precision (**1.0d0**).

Note: If annular sampling is used, the coordinates have to be selected as well.

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 they are normalised 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 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

 BEAM	Beam: Momentum ▼	p:	Part: ▼
Δp: Flat ▼	Δp:	Δφ: Flat ▼	Δφ:
Shape(X): Rectangular ▼	Δx:	Shape(Y): Rectangular ▼	Δy:
 BEAMPOS	x:	y:	z:
	cosX:	cosy:	Type: POSITIVE ▼
 BEAMAXES	cosBxx:	cosBxy:	cosBxz:
	cosBzx:	cosBzy:	cosBzz:

- 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:

- Particle type
- Energy / momentum
- Coordinates
- Direction
- Weight

```
source_newgen.f - Debug output
-----
Particle -E [GeV]      X [cm]      Y [cm]      Z [cm]      Cosx      Cosy      Cosz      Weight
3 -4.6846462E-02      0.0000000E+00  0.0000000E+00  0.0000000E+00 -1.2028663E-03 -4.0994198E-04  9.9999919E-01  1.0000000E+00
3 -2.2732349E-03      0.0000000E+00  0.0000000E+00  0.0000000E+00 -8.1620103E-04  5.0567202E-04  9.9999954E-01  1.0000000E+00
3 -3.0508784E-04      0.0000000E+00  0.0000000E+00  0.0000000E+00  4.6318357E-04  6.8805110E-04  9.9999966E-01  1.0000000E+00
3 -6.0162525E-04      0.0000000E+00  0.0000000E+00  0.0000000E+00 -8.4087805E-04  8.2140424E-04  9.9999931E-01  1.0000000E+00
3 -1.1779098E-03      0.0000000E+00  0.0000000E+00  0.0000000E+00  8.7684219E-04  1.5051092E-03  9.9999848E-01  1.0000000E+00
3 -2.1253372E-03      0.0000000E+00  0.0000000E+00  0.0000000E+00  4.8552257E-04 -1.3290855E-04  9.999987E-01  1.0000000E+00
3 -6.2072769E-04      0.0000000E+00  0.0000000E+00  0.0000000E+00  3.0720252E-04  8.0856146E-05  0.9999902E-01  1.0000000E+00
```

- 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:
```

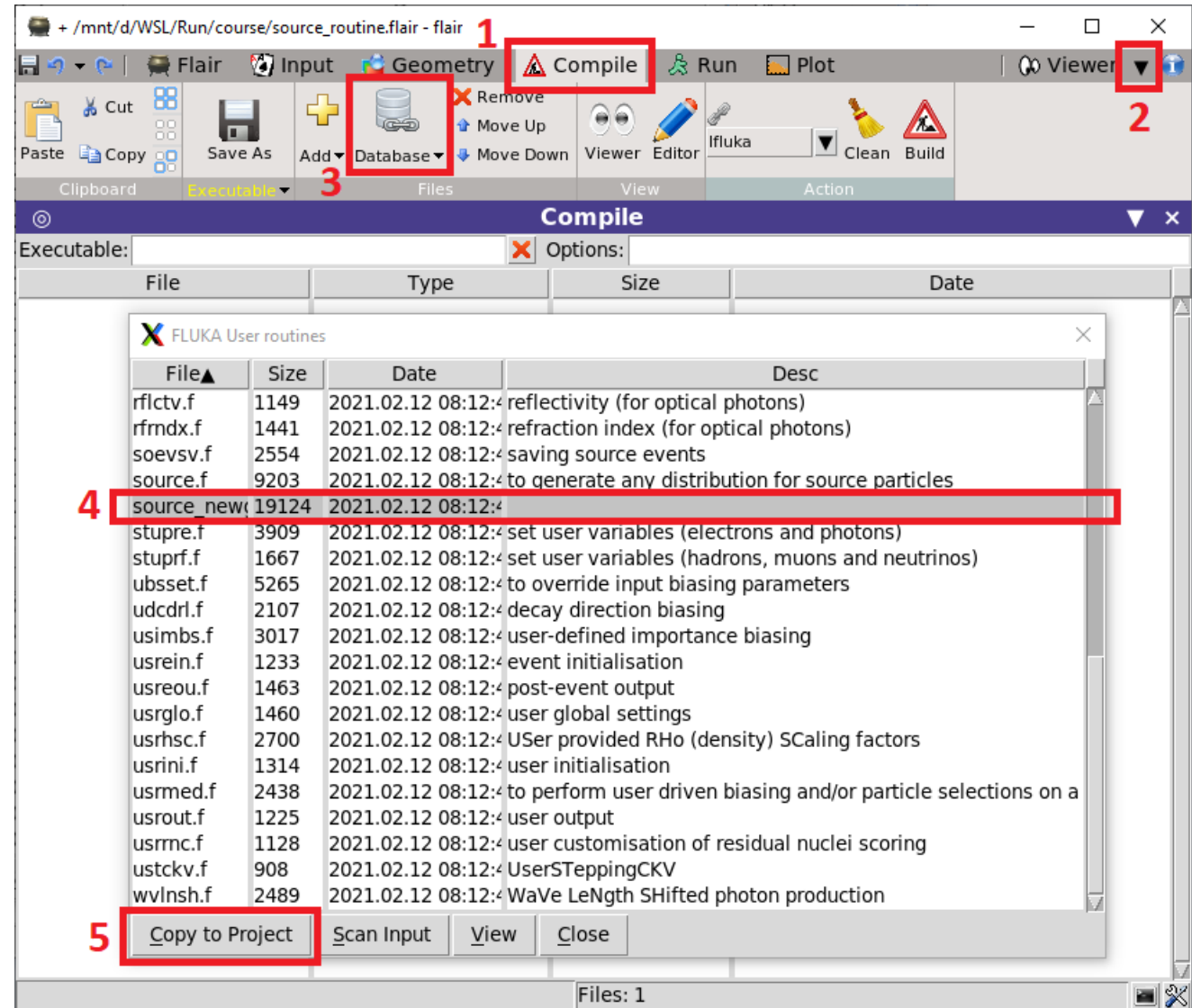
- Good practice:

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

Adding the user routine to the project folder

1. Open the [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]

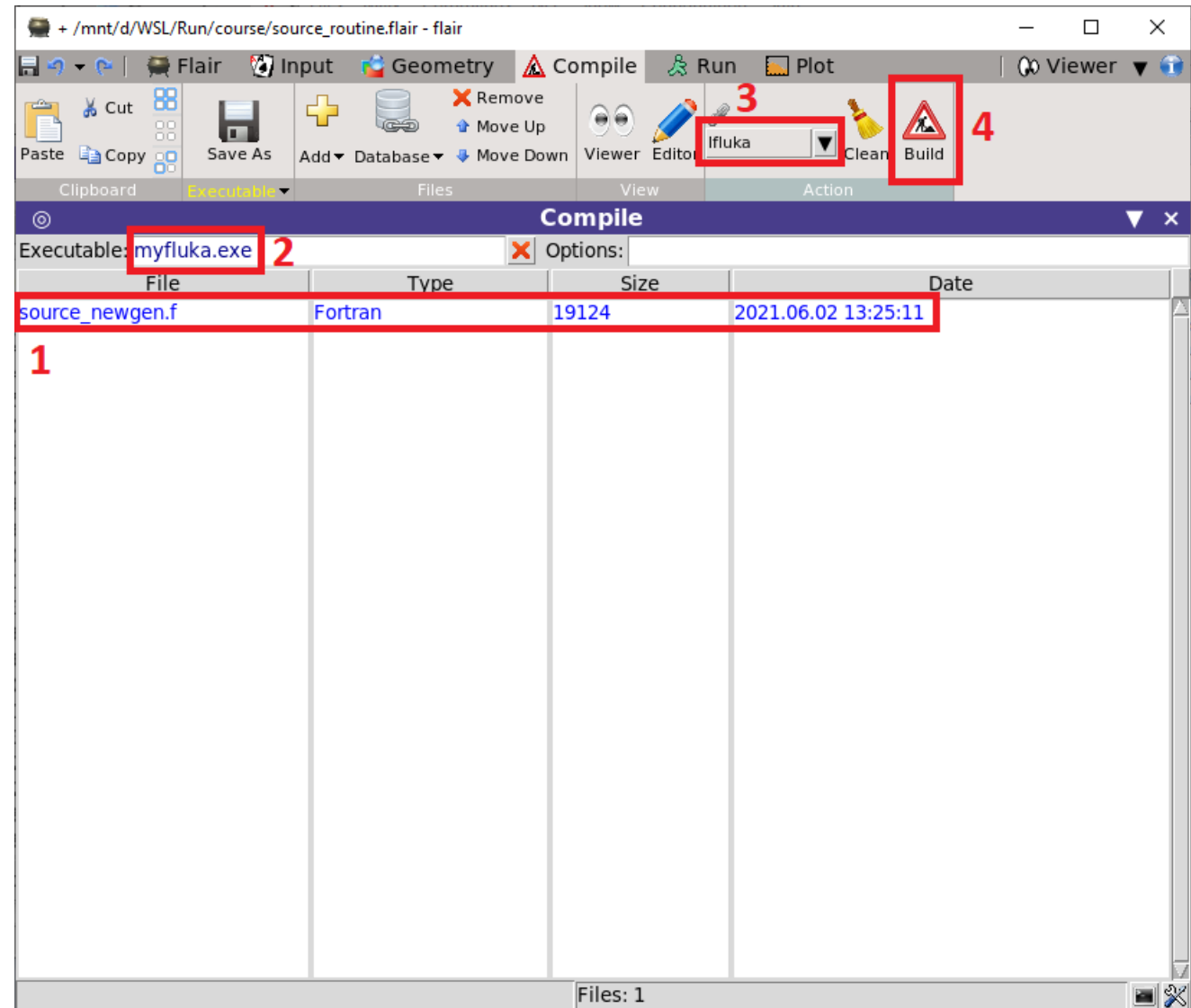
The copied user routine will be in the Flair project directory



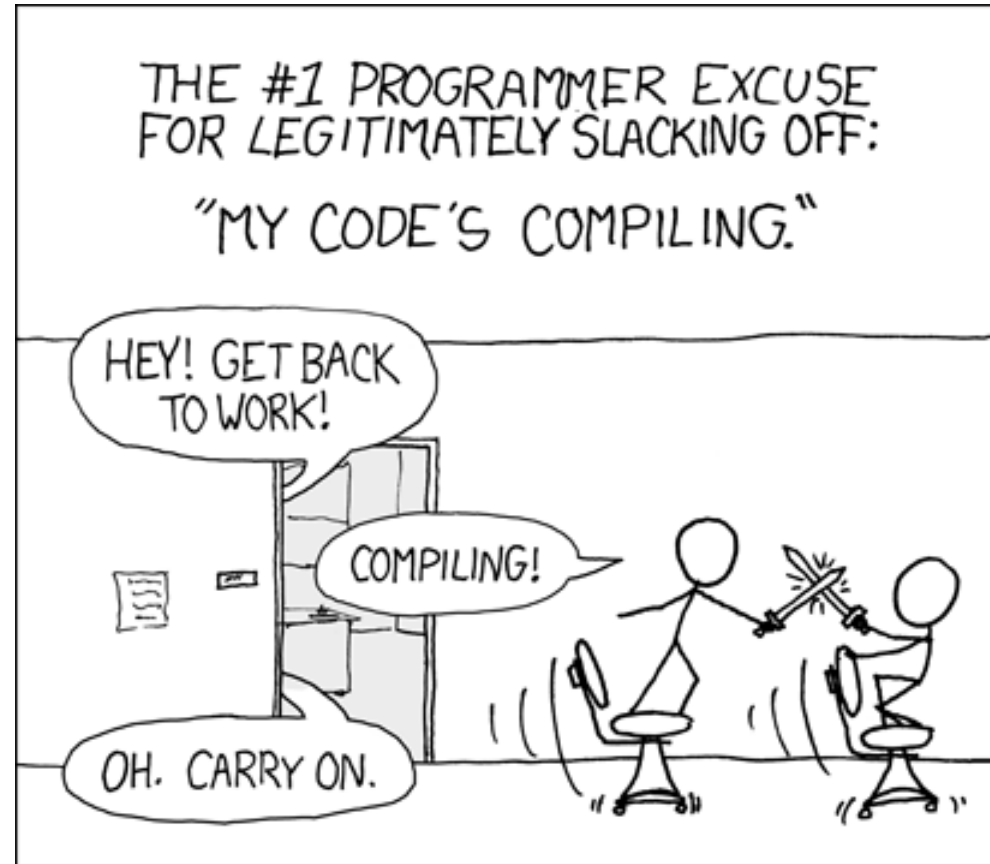
Compiling a custom FLUKA executable

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

Users must check that the custom executable is selected in the [Run] tab



Time for an exercise!



xkcd.com/303

