Explain the following sample codes:

(1) udm_int_sample1.f90

(2) udm_int_sample2.f90

(3) udm_part_sample1.f90

(4) udm_part_sample2.f90

(5) udm_Manager.f90

You can create your own user code by modifying the parts described below.

File name rules

The file name defining interactions should be as follows

```
udm_int_<Your File Name>.f90
```

The file name defining particles should be as follows

```
udm_part_<Your File Name>.f90
```

[Note]: You can name files other than these by writing directly to the makefile.

Description of the ***interaction file*** (*udm_int_*.f90*)

Two functions/subroutines that users primarily modify:

`function Xsec_per_atom(...)`

[Role]: Defines the total cross-section of the interaction.

`subroutine generate_final_state`

[Role]: Sampling final state energies, scattering angles, etc.

Description of the ***particle file*** (*udm_part_*.f90*)

Three functions/subroutines that users primarily modify:

`function mass()`

[Role]: Define the mass of a particle.

`function lifetime()`

[Role]: Define the lifetime.

`subroutine decay`

[Role]: Define the decay patern.

(1) udm_int_sample1.f90

An example of an interaction in which a user-defined particle (X) is emitted by an incident electron or positron.

e + atom (Z,A) → e + X + ...

```
1   ! Interaction Template Version = 1.0
2
3   !=======================================================================
4   module udm_int_sample_1
5   !=======================================================================
6
7   use udm_Parameter
8   use udm_Utility
9   implicit none
10  private ! Functions and variables are set to private by default.
11  public :: caller ! The 'caller' subroutine should be
12
13  !-----------------------------------------------------------------------
14  ! Default variables
15  !-----------------------------------------------------------------------
16  character(len=99), parameter :: Name = "my_interaction_1" !
17  double precision, allocatable, save :: Parameters(:) ! Para
18  integer, parameter :: num_initial = 2 ! The number of incid
19  integer, save :: kf_initial(num_initial) = (/ 11, -11 /) ! The kf
20  !-----------------------------------------------------------------------
21  ! User variables
22  !-----------------------------------------------------------------------
23  ! integer i,j
24  ! double precision x,y
25  integer kf_X
26  contains
```

Defines the module name. This is used in `udm_Manager.f90` described below.

Define a `Name` for this interaction. This is used in the input file.

The number of incident particles causing this interaction.

`kf-code` of the incident particle causing this interaction.
In this case, the electron (11) and positron (-11)

```
329
330  !=======================================================================
331  end module udm_int_sample_1
332  !=======================================================================
333
334
```

(1) udm_int_sample1.f90

```
39  !=======================================================================
40  subroutine initialize
41  ! This subroutine is called only once at the beginning of the calculation.
42  !=======================================================================
43  kf_X=Parameters(1) ! kf-codes (particle ID) of X (outgoing particle).
44  end subroutine initialize
```

Parameters entered in the `[user defined interaction]` section of the input file are automatically assigned to the array `Parameters(i)` in the source code.

(Example of input file)

```
[ user defined interaction ]
  n_int = 2

$ Name                  Bias     Parameters
  my_interaction_1      1        900000
  my_interaction_3      100      900000   1.1   2.2
```

This value is available as `Parameters(1)` in the source code whose `Name` is `my_interaction_1`.

`Parameters(1)`

`Parameters(2)`

`Parameters(3)`

(In `my_interaction_3`)

(1) udm_int_sample1.f90

Define the total cross section per atom for the function `Xsec_per_atom`. Units are in barn.

The kf-code of the incident particle is assigned to `udm_kf_incident` in the code

```
48  !=======================================================================
49  double precision function Xsec_per_atom(Kin,Z,A)
50  ! Integrated cross section per an atom.
51  ! Unit: barn (10^-24 cm2)
52  implicit none
53  double precision Kin ! Kinetic energy of incident particle [MeV]
54                   integer Z ! Atomic number of target atom
55                   integer A ! Mass   number of target atom
56  !-----------------------------------------------------------------------
57  ! [Variables available in this function]
58  ! udm_kf_incident : The kf-codes (particle IDs) of the incident particles.
59  !=======================================================================
60
61  if(Kin < 100.0) then
62    Xsec_per_atom=0.0
63    return
64  endif
65
66  if     (udm_kf_incident ==  11) then
67    Xsec_per_atom=1e-6*Z
68  else if(udm_kf_incident == -11) then
69    Xsec_per_atom=2e-6*Z
70  else
71    print*,"error"
72    stop
73  endif
74
75  return
76  end
```

In this case, the cross section is 0 barn if the incident particle kinetic energy (= Kin) is less than 100 MeV.

(10^{-6} * Z) barn, for electron.
($2*10^{-6}$ * Z) barn, for positron.

(1) udm_int_sample1.f90

First half

```
112  !=======================================================
113  subroutine generate_final_state
114  !=======================================================
115  ! Subroutine to determine final state infor
116  ! In this example, the X and e- momenta of
117  !-------------------------------------------------------
118  ! [Variables available in this subroutine]
119  ! udm_Kin          : Kinetic Energy of the particle
120  ! udm_kf_incident  : PDG code of the incident particle
121  !-------------------------------------------------------
122  integer Z_A_hit(2), Z, A
123  integer i, n_sampling_max
124  double precision Kout, Kout_min, Kout_max
125  double precision P, R
126  double precision m_X, E_X, p_X, theta_X
127  double precision m_e, E_e, p_e, theta_e
128  !-------------------------------------------------------
129  ! You may use Z and A for final state varia
130  ! Z and A of a target material are automati
131  Z_A_hit=get_hit_nuclide_Z_A(udm_Kin)
132  Z=Z_A_hit(1)
133  A=Z_A_hit(2)
134  !-------------------------------------------------------
135
136
137  ! ==========================================
138  ! =  Your original sampling section        =
139  ! ==========================================
140  ! Range of sampling variable
141  Kout_min=0.0d0
142  Kout_max=udm_Kin-get_mass(kf_X)
143  !-------------------------------------------------------
144  ! Start sampling
145  n_sampling_max = 100000
146  do i=1, n_sampling_max
147  !-------------------------------------------------------
148  ! Sampling of Kout in the range.
149  Kout=get_random(Kout_min, Kout_max)
150  !-------------------------------------------------------
151  ! Rejection sampling method.
152  P=distribution(udm_Kin, Z, A, Kout) ! The
153  R=get_random_0to1()                 ! Ra
154  ! If P > R, thi
155  if(P > R) then
156  !-------------------------------------------------------
157  ! Accepted!!!
158  ! Initialization is required before fi
159  call initialize_udm_event_info
160  !-------------------------------------------------------
161
162  !-------------------------------------------------------
163  ! Set number of final state
164  set_final_state_number = 2
165  !-------------------------------------------------------
```

Final states are sampled by `generate_final_state`. *Rejection sampling method* is used here.

Obtain interacted atom information (Z, A) if needed

The distribution of the emission energy (Kout) of the particle X is defined by another function.

This Kout is accepted

[Important]: Initialize the array before setting the final state information.

[Important]: Specify the number of final states

Second half

```
166
167  !-------------------------------------------------------
168  ! [Important Notice]
169  ! Here, you set the final state momenta to the following "set_*" array,
170  ! assuming the direction of the momentum of the incident particle to be
171  ! positive along the "Z-axis". The final state momenta are automatically
172  ! rotated based on the actual direction outside of this subroutine.
173  !-------------------------------------------------------
174
175  m_X=get_mass(kf_X)
176  E_X=Kout+m_X
177  p_X=sqrt(E_X**2-m_X**2)
178  theta_X=get_random(0.0d0,0.1d0)
179  !-------------------------------------------------------
180  ! Set 4-momentum of X
181  set_kf                    (1) = kf_X
182  set_Total_Energy_in_MeV(1) = E_X
183  set_Px_in_MeV             (1) = p_X*sin(theta_X)
184  set_Py_in_MeV             (1) = 0.0d0
185  set_Pz_in_MeV             (1) = p_X*cos(theta_X)
186
187  m_e=get_mass(udm_kf_incident)
188  E_e=udm_Kin+m_e-E_X
189  p_e=sqrt(E_e**2-m_e**2)
190  theta_e=get_random(0.0d0,0.1d0)
191  !-------------------------------------------------------
192  ! Set 4-momentum of e+-
193  set_kf                    (2) = udm_kf_incident
194  set_Total_Energy_in_MeV(2) = E_e
195  set_Px_in_MeV             (2) = p_e*sin(theta_e)
196  set_Py_in_MeV             (2) = 0.0d0
197  set_Pz_in_MeV             (2) = p_e*cos(theta_e)
198  !-------------------------------------------------------
199
200  ! [Additional Quantities]
201  ! set_isomer_level             (?) = ?? ! (Default=0)   0: Not nuclear isom
202  ! set_excitation_energy_in_MeV(?) = ?? ! (Default=0.0) Excitation energy [
203
204  !-------------------------------------------------------
205  ! The final states are recorded.
206  call fill_final_state
207  !-------------------------------------------------------
208  return
```

Assuming that the direction of the incident particle in the starting state is the Z-axis, the momentum of the end state is calculated.

Assume the direction of the incident particle is the Z-axis, and assign the kf-code, energy, and momenta of the 1st final state.

In this example, we assign the information for the 2nd final state.

[Important]: Call `fill_final_state`, after the assignment.

(2) udm_int_sample2.f90

Code similar to udm_int_sample1.f90. However, the incident particles are muons.

μ + atom (Z,A) → μ + X + ...

```
16   character(len=99), parameter :: Name = "my_interaction_2"
17   double precision, allocatable, save :: Parameters(:) ! Par
18   integer, parameter :: num_initial = 2 ! The number of inci
19   integer, save :: kf_initial(num_initial) = (/ 13, -13 /) !
```

[Changes]
The incident particles are
μ- (13) and μ+ (-13).

(3) udm_part_sample1.f90

Define the particle X generated by "udm_int_sample1.f90".

```
1   ! Particle Template Version = 1.0
2
3   !=========================================================
4   module udm_part_sample_1
5   !=========================================================
6
7   use udm_Parameter
8   use udm_Utility
9   implicit none
10  private ! Functions and variables are set to p
11  public :: caller ! The 'caller' subroutine sh
12
13  !---------------------------------------------------------
14  ! Default variables
15  !---------------------------------------------------------
16  character(len=99), parameter :: Name = "my_particle_1"
```

Define an arbitrary module name.
This is used in udm_Manager.f90 described below.

Define a Name for this particle. This is used in the input file.

```
145 !=========================================================
146 end module udm_part_sample_1
147 !=========================================================
```

(3) udm_part_sample1.f90

```
46  !=======================================================================
47  double precision function mass() ! Unit: MeV
48  !=======================================================================
49  mass=50.0 ! 50 MeV
50  return
51  end
52
53  !=======================================================================
54  double precision function lifetime() ! Unit: Second
55  ! The value should be greater than 0.
56  !=======================================================================
57  lifetime=0.1e-9 ! 0.1 nano second
58  return
59  end
60
61  !=======================================================================
62  subroutine decay
63  !=======================================================================
64  ! [Warning] available
65  ! udm_Kin          : Kinetic Energy of the incident particle [MeV]
66  !-----------------------------------------------------------------------
67  if(0.5 > get_random_0to1()) then
68    call two_body_decay_uniform(12,12)       ! X -> 2 neutrinos (50%)
69  else
70    call three_body_decay_uniform(12,12,12) ! X -> 3 neutrinos (50%)
71  endif
72  end subroutine decay
```

Set mass to 50 MeV

Set lifetime to 0.1 x 10^{-9} second

Branching Ratio 50%

Get a random number in the range of 0 to 1

`decay` subroutine defines decay pattern

`two_body_decay_uniform (kf1, kf2)`
Subroutine for 2-body decay, where kf-code of the final state is kf1 and kf2.

`three_body_decay_uniform (kf1, kf2, kf3)`
Subroutine for 3-body decay, where kf-code of the final state is kf1, kf2, and kf3.

In this case, 50% decays into two electron neutrinos (kf=12) and 50% into three electron neutrinos (unphysical, but for simplicity)

(4) udm_part_sample2.f90

Code similar to udm_part_sample1.f90.

The difference is that the mass and lifetime use the values entered in the [user defined particle] section.

```
 4  module udm_part_sample_2
                    ⋮
16  character(len=99), parameter :: Name = "my_particle_2"
                    ⋮
46  !=============================================================
47  double precision function mass() ! Unit: MeV
48  !=============================================================
49  mass=Parameters(1) ! MeV
50  return
51  end
52
53  !=============================================================
54  double precision function lifetime() ! Unit: Second
55  ! The value should be greater than 0.
56  !=============================================================
57  lifetime=Parameters(2) ! second
58  return
59  end
```

(Example of input file)

```
[ user defined particle ]
  n_part = 1

$ Name            kfcode    Parameters
  my_particle_2   900000    100   1.0e-9
```

Parameters(1) Parameters(2)

(5) udm_Manager.f90

Write the module name of the user code you want to use in udm_Manager.f90.

Line up all modules you want to use.

```
1  module udm_Manager
2  use udm_Parameter
3
4  !============================================================
5  ! [udm_int]
6  use udm_int_sample_1,   caller_udm_int_sample_1  => caller
7  use udm_int_sample_2,   caller_udm_int_sample_2  => caller
8  ! [udm_part]
9  use udm_part_sample_1,  caller_udm_part_sample_1 => caller
10 use udm_part_sample_2,  caller_udm_part_sample_2 => caller
11 !============================================================
12
```

use <module name>, caller_<module name> => caller

```
15
16
17
18
19 subroutine user_defined_interaction(action,index)
20 integer action,index
21 !============================================================
22 call caller_udm_int_sample_1(action,index)
23 call caller_udm_int_sample_2(action,index)
24 !============================================================
25 end subroutine user_defined_interaction
```

Line up all interactions you want to use.

call caller_<module name>(action,index)

```
28
29
30 subroutine user_defined_particle(action)
31 integer action,index
32 do index=1,udm_part_nMax
33 !============================================================
34 call caller_udm_part_sample_1(action,index)
35 call caller_udm_part_sample_2(action,index)
36 !============================================================
37 enddo
```

Line up all particles you want to use.

call caller_<module name>(action,index)

```
40 end module udm_Manager
```