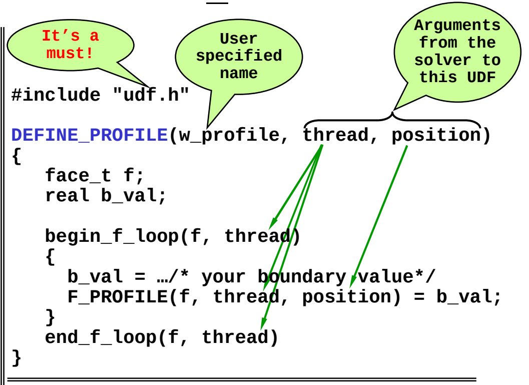
UDF Hooks --- ‘DEFINE’ Macros

Advanced UDF
Modeling Course

Boundary Profiles: DEFINE_PROFILE

- You can use this UDF to specify
  - Wall
    - temperature
    - heat flux, shear stress
  - Inlets
    - velocity
    - temperature
    - turbulence
    - species
    - scalars
- The macro **`begin_f_loop`** loops over all faces on the selected boundary thread
- The **`F_PROFILE`** macro applies the value to face, **`f`** on the **`thread`**



```
#include "udf.h"

DEFINE_PROFILE(w_profile, thread, position)
{
   face_t f;
   real b_val;

   begin_f_loop(f, thread)
   {
     b_val = …/* your boundary value*/
     F_PROFILE(f, thread, position) = b_val;
   }
   end_f_loop(f, thread)
}
```

`thread` : The thread of the boundary to which the profile is attached

`position` : A solver internal variable (identifies the stack location of the profile in the data stack)

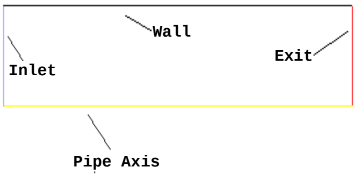
User can rename the variables at will:
`DEFINE_PROFILE(my_prof, t, pos)`

Example 1: Transient Inlet Velocity

- Pulsatile flow in a tube

  $V_x = V_o + A \sin(\omega t)$

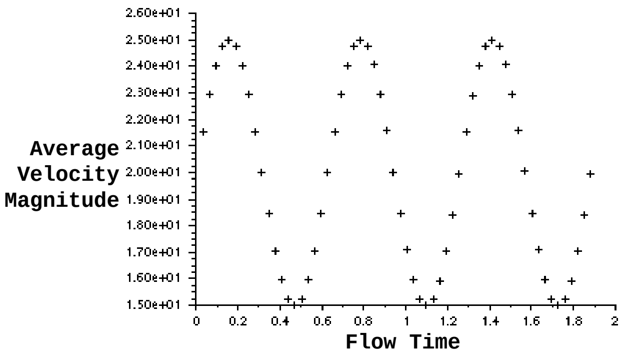
  where V_o = 20 m/s, A = 5 m/s, ω = 10 rad/s
- Boundary condition is applied at inlet



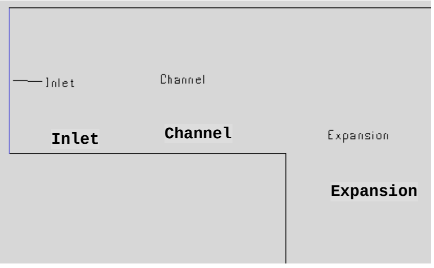
```
#include "udf.h"
DEFINE_PROFILE(unsteady_v, t, pos)
{
  real time, velocity;
  face_t f;
  begin_f_loop(f, t)
    {
     time = RP_Get_Real("flow-time");
     velocity = 20.0 +
        5.0*sin(10.*time);
     F_PROFILE(f, t, pos) = velocity;
    }
  end_f_loop(f, t)
}
```

Example 1: Results of Transient Inlet Velocity

- Time history of the average velocity at the pipe exit shows sinusoidal oscillation with a mean of 20 and amplitude of 5.



Example 2: Fully Developed Turbulent Inlet

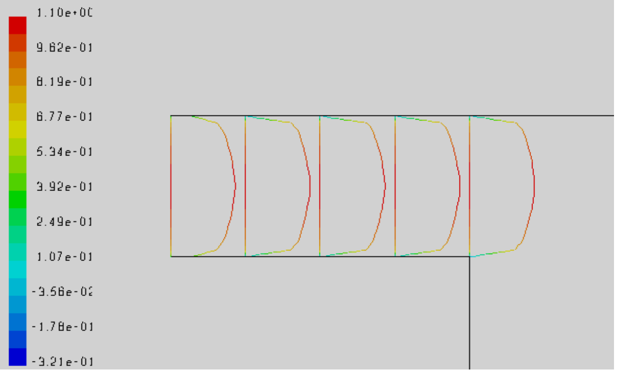


- Profiles for inlet velocity, **k** and **ε** are used to approximate fully developed flow conditions
- Velocity profile follows 1/7 power law
- Turbulent kinetic energy varies linearly from a near-wall peak to a prescribed core-flow value
- Dissipation is prescribed by a mixing-length model
- Used to minimize the domain size and sensitivity to inlet boundary conditions

---

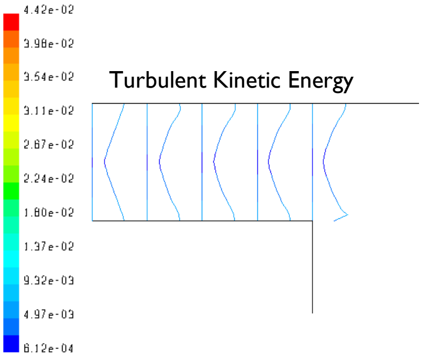
Example 2: Results of Fully Developed Turbulent Inlet

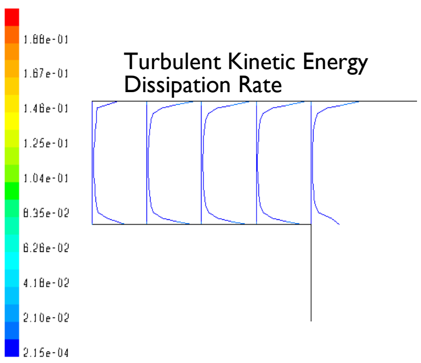
- Axial velocity profile changes little downstream of inlet boundary



Example 2: Results of Fully Developed Inlet

- Turbulence quantities change little downstream of the inlet


Turbulent Kinetic Energy

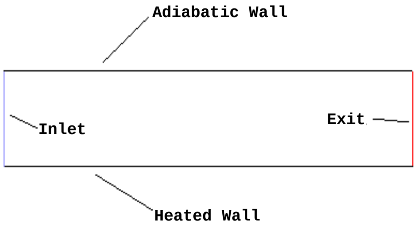

Turbulent Kinetic Energy Dissipation Rate

Example 3: Sinusoidal Wall Temperature

- Lower wall temperature varies sinusoidally with x-position according to

$$T_x = 300 + 100 \sin(\pi x/L)$$

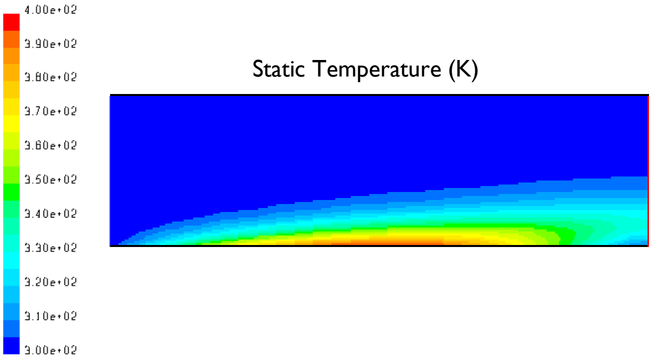
- Inlet fluid enters at 300 K
- Upper wall is insulated



Temperature:	`F_PROFILE(f, t, pos) = 300.+100.*sin(PI*x/0.005);`

Example 3: Results of Sinusoidal Wall Temperature

- Wall (and fluid) temperature reaches peak at midlength of channel



Source Terms (1)

- The solvers compute source terms using the “linearized form”

$$S = A + B\,\phi$$

where ϕ is the dependent variable, A is the explicit part of the source term and $B\phi$ is the implicit part

- A recommended linearization is $S = S^* + \left(\frac{\partial S}{\partial \phi}\right)^* (\phi - \phi^*)$
where ϕ is the dependent variable

- FLUENT Solver will automatically determine whether the user-supplied source is enhancing the numerical stability (namely, the diagonal dominance of the system matrix)

Source Terms (2)

- Source term UDFs can be created for the governing equations:
  - continuity
  - momentum
  - k, ε
  - energy
  - species
  - User-defined scalars
- Energy source term UDFs may also be defined for solid zones
- NOTE: The units of all source terms are expressed in terms of the volumetric generation rate. For example, a source term for the continuity equation would have units of (kg/s/m^3)

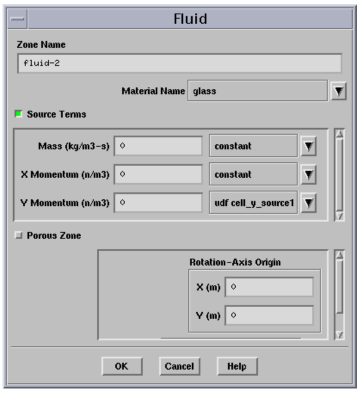
Source Terms (3)

- Solver call this UDF for each cell in the zone
- The solver passes the UDF the cell pointer associated with the cell
- The variable **`dS[eqn]`** sets up the implicit part of the source term for the equation the source term is used for
- Note that the UDF returns a real value for the explicit part of the source, the implicit part dS[eqn] is returned in a referenced array

```
include "udf.h"

DEFINE_SOURCE(cell_y_source1,
       cell, thread, dS, eqn)
{

real source;

/* S = source + dS[eqn]*phi */

dS[eqn] =  /* expression */

source =   /* expression */

 return source;
}
```

Source Terms (4)

- To activate source terms **`Define⌫Boundary Conditions⌫fluid-1`** and click on **`Source Terms`**



Fluid

Zone Name: fluid-2

Material Name: glass

☑ Source Terms

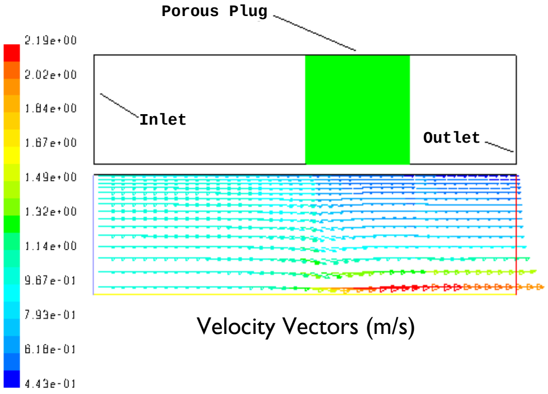
Mass (kg/m3-s): 0 — constant
X Momentum (n/m3): 0 — constant
Y Momentum (n/m3): 0 — udf cell_y_source1

☐ Porous Zone

Rotation-Axis Origin
X (m): 0
Y (m): 0

Example 4: Position Dependent Porous Media

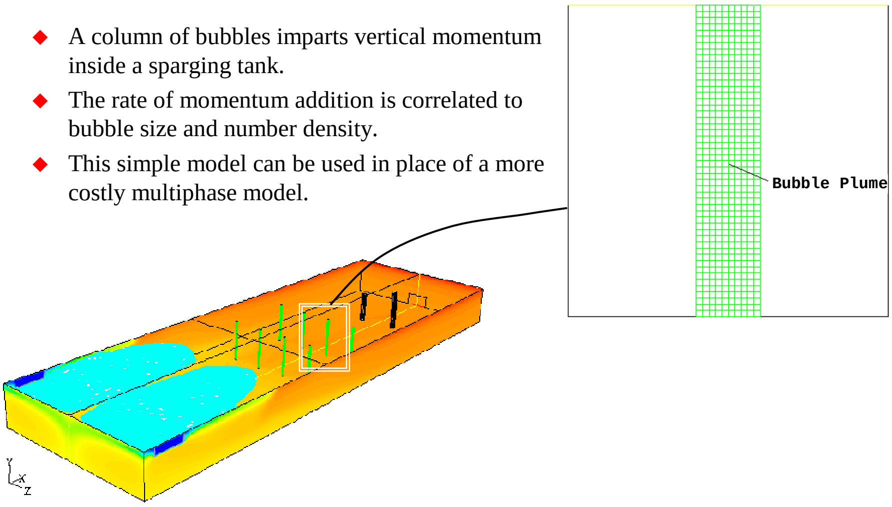
- Channel flow with porous plug
- x-momentum loss is linear in y-position, starting from zero at lower wall
- Fluid flows preferentially near the bottom of the channel



Velocity Vectors (m/s)

Example 5: Bubble Generated Momentum

- A column of bubbles imparts vertical momentum inside a sparging tank.
- The rate of momentum addition is correlated to bubble size and number density.
- This simple model can be used in place of a more costly multiphase model.

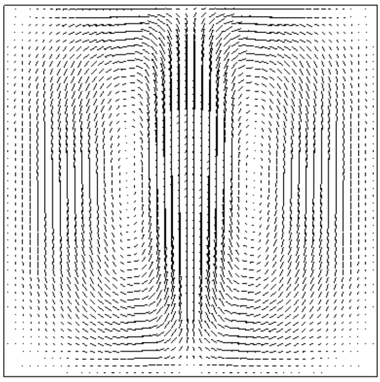


Example 5: Bubble Generated Momentum

- The rising plume of bubbles creates circulation throughout the tank

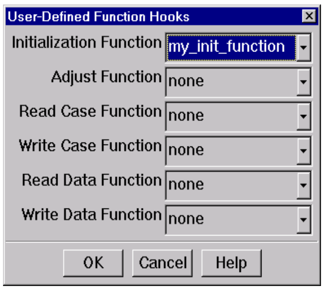
$v=g*r^2*\rho/(3*\mu)$
$Drag=4*\pi*\mu*r*v$
$N = f*h/v$
$Source=N*Drag*100/Volume$

```
#include "udf.h"
real bubbler_vol=0.;/*static variable*/
DEFINE_SOURCE(mom_y_src, c, t, rj, eqn)
{
#define PI 3.14159
#define GRAV 9.81
#define bub_rad 1.e-3
real bub_vel,f_d,bub_freq=5.,bubbler_ht=1.;
float bub_num, source;
cell_t cc;
rj[eqn] = 0.0;
if(bubbler_vol == 0.) /*Bubbler volume*/
{begin_c_loop(cc, t)
 bubbler_vol=bubbler_vol+C_VOLUME(cc,t);
 end_c_loop(cc, t)}
/* Calculate force for single bubble */
bub_vel=GRAV*pow(bub_rad,2.)*C_R(c,t)/
                      (3.*C_MU_L(c,t));
f_d =4.*PI*C_MU_L(c,t)*bub_rad*bub_vel;
bub_num = (bub_freq*bubbler_ht/bub_vel);
source = bub_num*f_d*100./bubbler_vol;
return source;
}
```



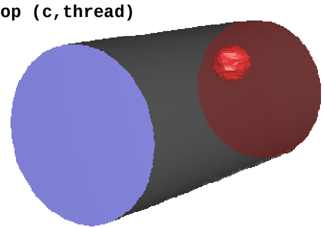
Initialization and Example 6

- Initializes solutions for entire domain, similar to “patching” of values
- Executed once at the beginning of solution process
- Initializatio Function appears under **`Define➔ User_Defined➔ Function_hooks…`**



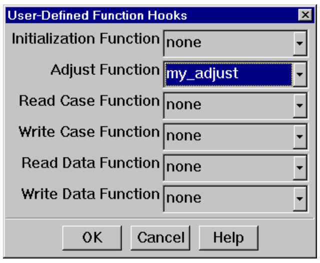
Initialization Function: my_init_function
Adjust Function: none
Read Case Function: none
Write Case Function: none
Read Data Function: none
Write Data Function: none

```
#include "udf.h"
DEFINE_INIT(my_init_function,domain)
{
  cell_t c;
  Thread *thread;
  real xc[ND_ND];
  thread_loop_c (thread,domain)
 {
      begin_c_loop (c,thread)
    {
      C_CENTROID(xc,c,thread);
if (sqrt(ND_SUM(pow(xc[0]-0.5,2.),
     pow(xc[1] - 0.5,2.),
     pow(xc[2] - 0.5,2.))) < 0.25)
    C_T(c,thread) = 400.;
else
      C_T(c,thread) = 300.;
    }
      end_c_loop (c,thread)
 }
}
```



Adjust Function and Example 7

- Function called for every iteration
- Integrate the turbulent dissipation over the whole domain and print it to the text user interface
- Adjust Function appears under **`Define➔User_Defined➔ Function_hooks…`**

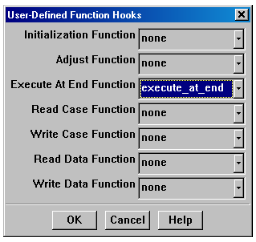


Initialization Function: none
Adjust Function: my_adjust
Read Case Function: none
Write Case Function: none
Read Data Function: none
Write Data Function: none

```
DEFINE_ADJUST(my_adjust, domain)
{
/* Integrate dissipation. */
 real sum_diss=0.;
 Thread *t;
 cell_t c;
 thread_loop_c (t,domain)
 {
  begin_c_loop (c,t)
  sum_diss += C_D(c,t)* C_VOLUME(c,t);
  end_c_loop (c,t)
 }
 Message(“Volume integral of turbulent
   dissipation : %g\n”,sum_diss);
}
```

Execute_at_End Function and Example 8

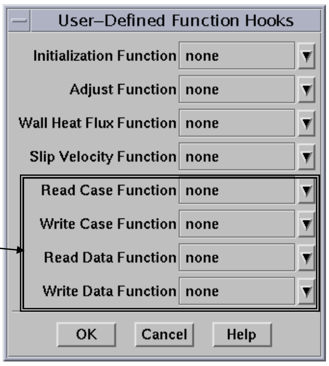
- This is a general purpose macro executed at the end of
  - an iteration in a steady state run, or
  - at the end of a time step in a transient run.
- UDF for integrating turbulent dissipation and printing it to console window at the end of the current iteration or time step
- This Function appears under `Define➔User_Defined➔Function_hooks…`


User-Defined Function Hooks
Initialization Function: none
Adjust Function: none
Execute At End Function: execute_at_end
Read Case Function: none
Write Case Function: none
Read Data Function: none
Write Data Function: none

```
#include "udf.h"
DEFINE_EXECUTE_AT_END(execute_at_end)
{ Domain *d; Thread *t;
  real sum_diss=0.;
  cell_t c;
  d = Get_Domain(1);
  thread_loop_c (t,d)
  { if (FLUID_THREAD_P(t))
    { begin_c_loop (c,t)
    sum_diss+=C_D(c,t)*C_VOLUME(c,t);
  end_c_loop (c,t)
  }
}
printf("Volume integral of turbulent
   dissipation: %g\n", sum_diss);
   fflush(stdout); }
```

User Defined I/O

- Ability to read/write custom data in case/data files
  - Can save and restore custom variables of any data types (e.g., integer, real, Boolean, structure)
  - Useful to save "***dynamic***" information (e.g., number of occurrences in conditional sampling)
  - Defined using **`DEFINE_RW_FILE`** macro
  - Selected in the User-Defined Function Hooks panel


User–Defined Function Hooks
Initialization Function: none
Adjust Function: none
Wall Heat Flux Function: none
Slip Velocity Function: none
Read Case Function: none
Write Case Function: none
Read Data Function: none
Write Data Function: none

User Defined I/O (2)

```
#include "udf.h"
int count = 0;  /* define and initialize static variable
   count */
DEFINE_ADJUST(it_count, domain)
{
  count++;
  printf("count = %d\n",count);
}
DEFINE_RW_FILE(writer, fp)
{
  printf("Writing UDF data to data file...\n");
  fprintf(fp, "%d",count); /* write out count to data
   file */
}
DEFINE_RW_FILE(reader, fp)
{
  printf("Reading UDF data from data file...\n");
  fscanf(fp, "%d",&count); /* read count from data file
   */
}
```

Properties and Example 9

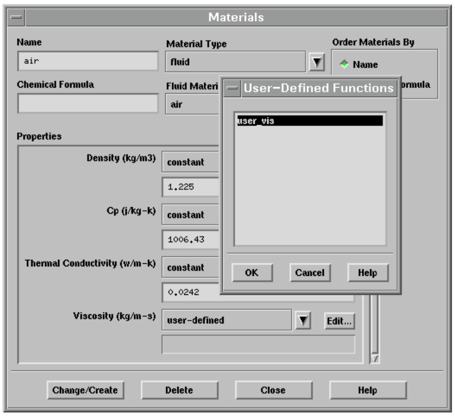
- UDF’s can be used to define
  - Viscosity
  - Thermal Conductivity
  - Mass Diffusivity
  - Density
- UDF’s cannot be used to define specific heat
- The function is called for every cell in the zone

$$\mu = \begin{cases} 5.5\,10^{-3} & \mathrm{T} > 288\mathrm{K} \\ 143.2 - 0.49725\mathrm{T} & 286\mathrm{K} \le \mathrm{T} \le 288\,\mathrm{K} \\ 1 & \mathrm{T} < 286\mathrm{K} \end{cases}$$

```
#include "udf.h"
DEFINE_PROPERTY(user_vis, cell, thread)
{
  real temp, mu_lam;
  temp = C_T(cell, thread);
  {
   if (temp > 288.)
     mu_lam = 5.5e-3;
   else if (temp >= 286.&& temp<=288.)
     mu_lam = 143.2135 - 0.49725 * temp;
    else
       mu_lam = 1.0;
  }
  return mu_lam;
}
```

Properties (2)

- To activate the UDF, select user-defined from the property drop down list
- When you select the user-defined option, a panel will appear with the names of your UDF's
- Select the name of the appropriate UDF

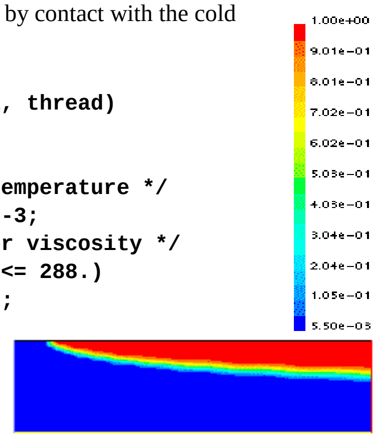


Name: air
Material Type: fluid
Chemical Formula:
Fluid Materi: air
Properties
Density (kg/m3): constant, 1.225
Cp (j/kg-k): constant, 1006.43
Thermal Conductivity (w/m-k): constant, 0.0242
Viscosity (kg/m-s): user-defined
User-Defined Functions: user_vis

Example 10: Temperature Dependent Viscosity

- Warm fluid enters the channel flowing from left to right.
- Viscosity increases as the fluid is cooled by contact with the cold upper wall.

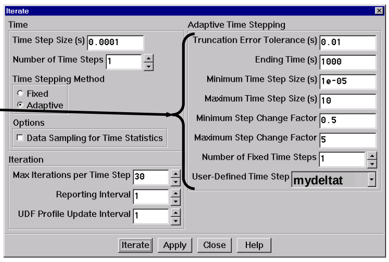
```
#include "udf.h"
DEFINE_PROPERTY(user_vis, cell, thread)
{real temp, mu_lam;
temp = C_T(cell, thread);
{/* Limit viscosity for high temperature */
if (temp > 288.) mu_lam = 5.5e-3;
/* Otherwise, use a profile for viscosity */
else if (temp >= 286. && temp <= 288.)
mu_lam = 143.2135-0.49725*temp;
else
mu_lam = 1.0;
}
return mu_lam;
}
```



Contours of molecular viscosity (kg/ms)

Time Step: DEFINE_DELTAT

- In Fluent, you may use adaptive timesteping based on minimum and maximum values of timesteps as well as other parameters
- Adaptive timestepping is activated by selecting the corresponding radio-button in the **`Solve-Iterate`** panel for unsteady problems
- **`DEFINE_DELTAT`** lets the user control the timestep based on any custom logic/algorithm


Iterate

Time
Time Step Size (s): 0.0001
Number of Time Steps: 1
Time Stepping Method: ○ Fixed ● Adaptive
Options: ☐ Data Sampling for Time Statistics

Iteration
Max Iterations per Time Step: 30
Reporting Interval: 1
UDF Profile Update Interval: 1

Adaptive Time Stepping
Truncation Error Tolerance (s): 0.01
Ending Time (s): 1000
Minimum Time Step Size (s): 1e-05
Maximum Time Step Size (s): 10
Minimum Step Change Factor: 0.5
Maximum Step Change Factor: 5
Number of Fixed Time Steps: 1
User-Defined Time Step: mydeltat

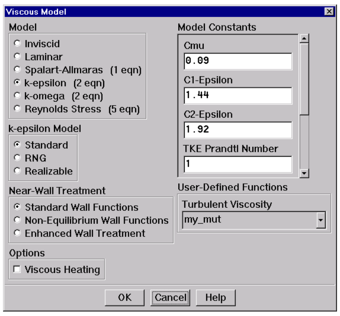
```
#include "udf.h"
DEFINE_DELTAT(mydeltat, domain)
{
real time_step;
real flow_time =
   RP_Get_Real("flow-time");
if (flow_time < 0.5)
   time_step = 0.1;
 else
   time_step = 0.2;
return time_step;
}
```

Turbulent Viscosity: DEFINE_TURBULENT_VISCOSITY

```
DEFINE_TURBULENT_VISCOSITY(my_mut,cell,thread)
{
 real mu_t;
 real rho = C_R(cell,thread);
 real k=C_K(cell,thread);
 real epsilon=C_D(cell,thread);
 mut= M_keCmu*rho*SQR(k)/epsilon;
 return mut;
}
```

$$\mu_t = C_\mu \rho \frac{k^2}{\varepsilon}$$

- Any custom relation for the turbulent viscosity formulation can be adopted using this UDF hook
- The variable names for the constants in the standard k-ε model are:
  - C_1 : **`M_keC1`**
  - C_2 : **`M_keC2`**
  - C_μ : **`M_keCmu`**
  - σ_k : **`M_keigk`**
  - σ_ε : **`M_keige`**
  - σ_ε : **`M_keprt`**


Viscous Model

Model: ○ Inviscid ○ Laminar ○ Spalart-Allmaras (1 eqn) ● k-epsilon (2 eqn) ○ k-omega (2 eqn) ○ Reynolds Stress (5 eqn)

k-epsilon Model: ● Standard ○ RNG ○ Realizable

Near-Wall Treatment: ● Standard Wall Functions ○ Non-Equilibrium Wall Functions ○ Enhanced Wall Treatment

Options: ☐ Viscous Heating

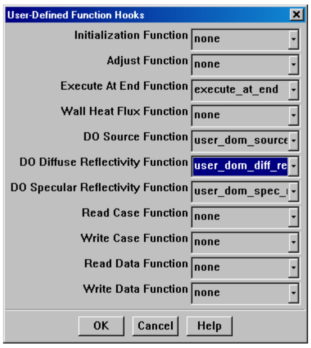
Model Constants
Cmu: 0.09
C1-Epsilon: 1.44
C2-Epsilon: 1.92
TKE Prandtl Number: 1

User-Defined Functions
Turbulent Viscosity: my_mut

Radiation Reflectivity: Discrete Ordinate Model Only

- Diffused **Reflectivity**
- Modify the interfacial reflectivity at diffusely reflecting semi-transparent walls, based on the refractive index
- This function is called for each semi-transparent wall and each band (non-gray DO Model)
- The function can be used to modify interface values of diffuse reflectivity and diffuse transmissivity
- In this example, reflectivity values are not customized: they are just printed

```
#include "udf.h"
DEFINE_DOM_DIFFUSE_REFLECTIVITY
(user_dom_diff_refl, t, nband,
 n_a, n_b, diff_ref_a, diff_tran_a,
 diff_ref_b, diff_tran_b)
{
 printf("diff_ref_a=%f diff_tran_a=%f\n",
        *diff_ref_a, *diff_tran_a);
 printf("diff_ref_b=%f diff_tran_b=%f \n",
        *diff_ref_b, *diff_tran_b);
}
```

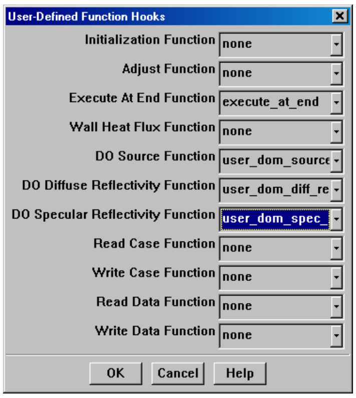


Initialization Function: none
Adjust Function: none
Execute At End Function: execute_at_end
Wall Heat Flux Function: none
DO Source Function: user_dom_source
DO Diffuse Reflectivity Function: user_dom_diff_re
DO Specular Reflectivity Function: user_dom_spec_(
Read Case Function: none
Write Case Function: none
Read Data Function: none
Write Data Function: none

Radiation Reflectivity: Discrete Ordinate Model (2)

- Specular **Reflectivity**
- Modify the specular reflectivity and transmitivity at semi-transparent walls, along direction s at a face (f)
- The same UDF is called for all the faces of the semi-transparent wall, for each of the directions

```
#include "udf.h"
DEFINE_DOM_SPECULAR_REFLECTIVITY
       (user_dom_spec_refl, f, t, nband, n_a,
n_b,
               ray_direction, en,
internal_reflection,
               specular_reflectivity,
specular_transmissivity)
{ real angle, cos_theta;
  real PI = 3.141592;
  cos_theta = NV_DOT(ray_direction, en);
  angle = acos(cos_theta);
  if (angle >45 && angle < 60)
     { *specular_reflectivity = 0.3;
       *specular_transmissivity = 0.7;
 } }
```

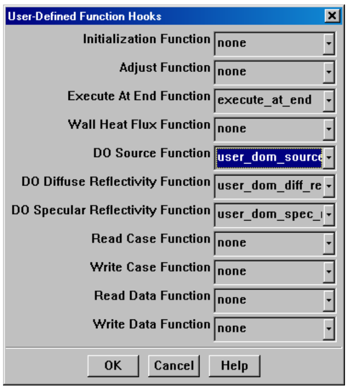


Initialization Function: none
Adjust Function: none
Execute At End Function: execute_at_end
Wall Heat Flux Function: none
DO Source Function: user_dom_source
DO Diffuse Reflectivity Function: user_dom_diff_re
DO Specular Reflectivity Function: user_dom_spec_
Read Case Function: none
Write Case Function: none
Read Data Function: none
Write Data Function: none

Emission & Scattering: Discrete Ordinate Source Macro

- Can be used to modify the emission and scattering terms in the radiative transport equation

```
#include "udf.h"
DEFINE_DOM_SOURCE(user_dom_source,
c, t, ni, nb, emission,
in_scattering, abs_coeff,
scat_coeff)
{
 *emission *= 1.05;
}
```

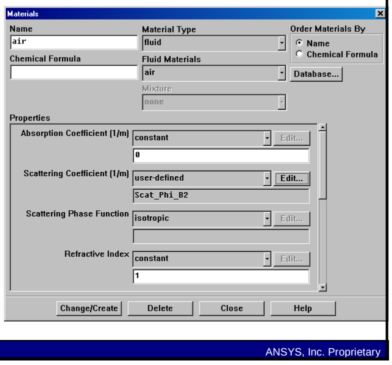


Initialization Function: none
Adjust Function: none
Execute At End Function: execute_at_end
Wall Heat Flux Function: none
DO Source Function: user_dom_sourc
DO Diffuse Reflectivity Function: user_dom_diff_re
DO Specular Reflectivity Function: user_dom_spec_
Read Case Function: none
Write Case Function: none
Read Data Function: none
Write Data Function: none

Scattering Phase Function: Discrete Ordinate Model

- Define the radiation scattering phase function for the Discrete Ordinates (DO) model
- The function computes two values: the fraction of radiation energy scattered from direction *i* to direction *j*, and the forward scattering factor
- Look at the UDF manual for a complete listing of the UDF for backward and forward sctteting phase functions after Jendoubi et al *J. Thermophys. Heat Transfer*, 7(2):213-219, 1993
- This function is loaded as user-defined scattering coefficient in the materials panel

```
#include "udf.h"
DEFINE_SCAT_PHASE_FUNC(Scat_Phi_B2,c,fsf)
{
 real phi=0;
 *fsf = 0;
 phi = 1.0 - 1.2*c + 0.25*(3*c*c-1);
 return (phi);
}
```



Name: air
Chemical Formula:
Material Type: fluid
Fluid Materials: air
Mixture: none
Order Materials By: ● Name ○ Chemical Formula
Absorption Coefficient (1/m): constant, 0
Scattering Coefficient (1/m): user-defined, Scat_Phi_B2
Scattering Phase Function: isotropic
Refractive Index: constant, 1

Additional Macros

- There are a number of additional model specific macros
  - You can learn more about these from the UDF manual section 4.3

```
DEFINE_CHEM_STEP( name, ifail, n, dt, p, temp, yk)
DEFINE_NET_REACTION_RATE( name, p, temp, yi, rr, jac)
DEFINE_NOX_RATE ( name, c, t, NOx)
DEFINE_PRANDTL_D ( name, c, t)
DEFINE_PR_RATE ( name, c, t, r, mw, ci, p, sf,
                 dif_index,   cat_index, rr)
DEFINE_SR_RATE ( name, f, t, r, my, yi, rr)
DEFINE_VR_RATE ( name, c, t, r, mw, yi, rr, rr_t)
DEFINE_TURB_PREMIX_SOURCE ( name, c, t, turb_flame_speed,
                            source)
```

- Multiphase specific macros will be discussed later