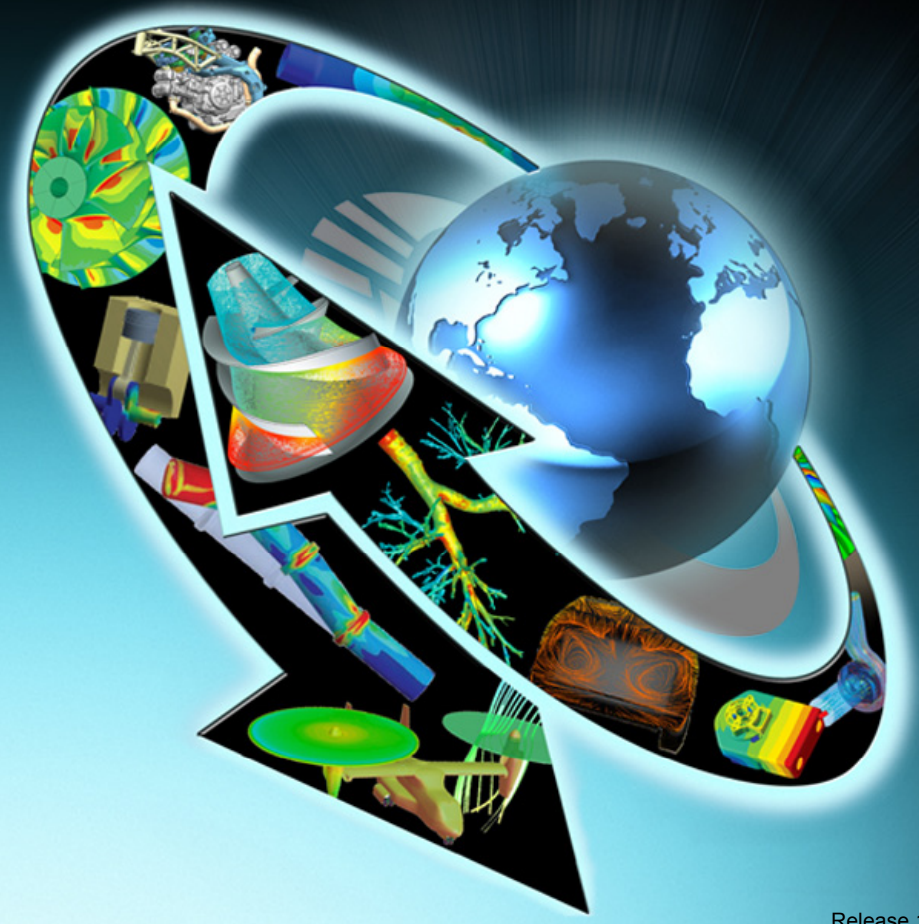
ANSYS®

Chapter 3

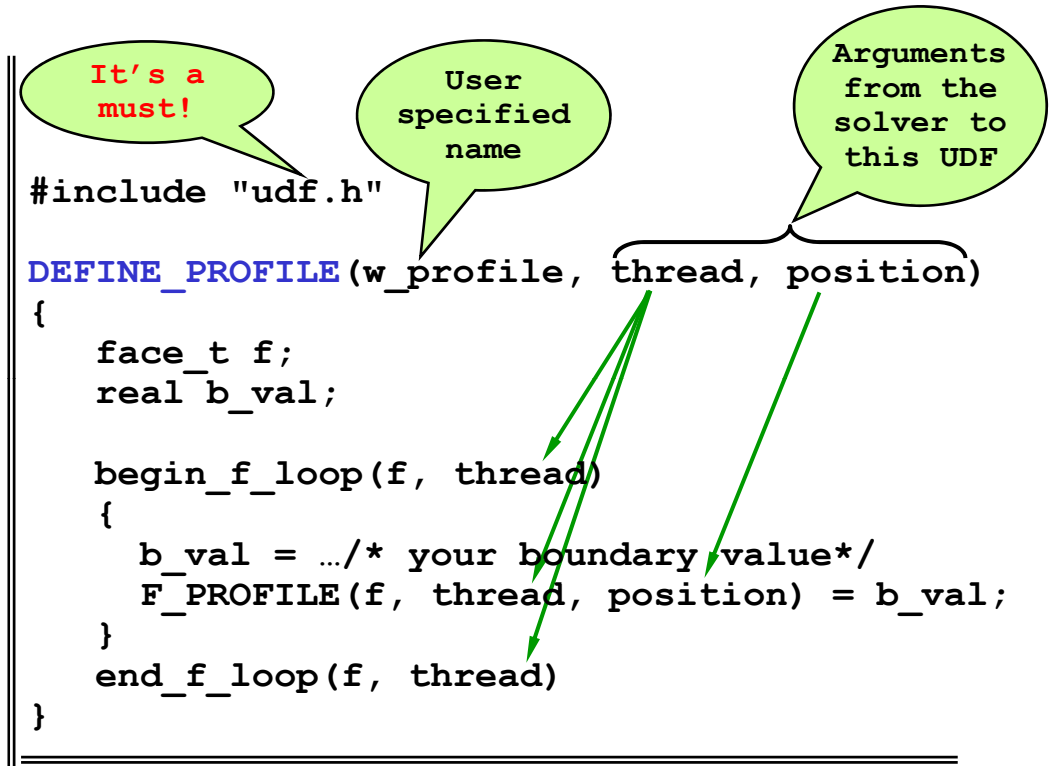
Top-Level ‘DEFINE’ Macros



Advanced FLUENT User-Defined Functions

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**

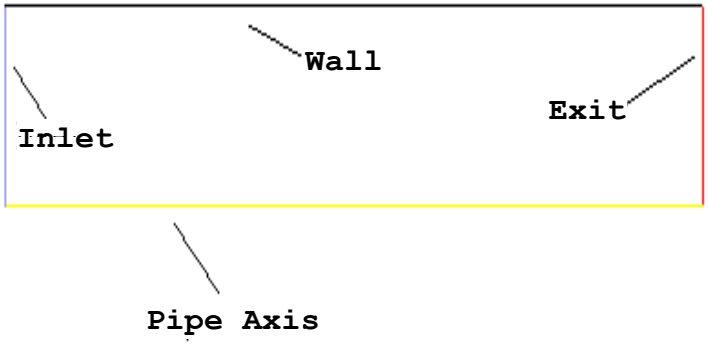


Example 1: Transient Inlet Velocity

- Pulsatile flow in a tube

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

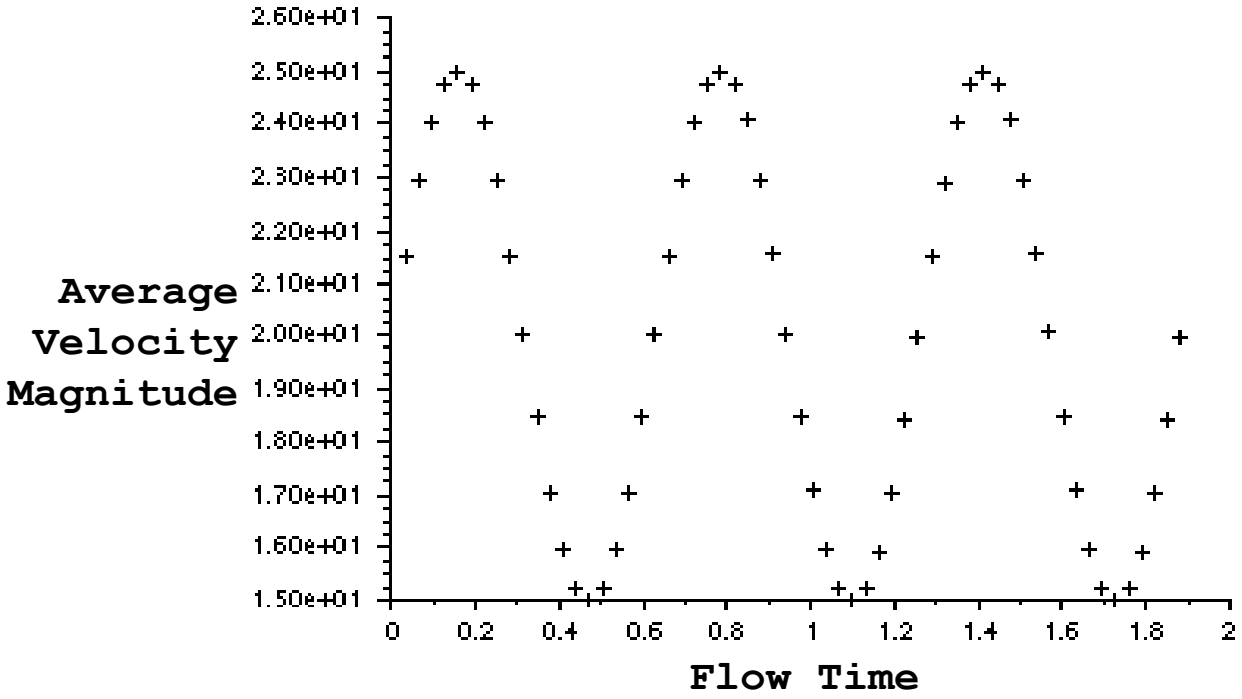
  where V_o = 20 m/s, A = 5 m/s, ω = 10 rad/s
- Boundary condition is applied at inlet




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

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 m/s and amplitude of 5 m/s

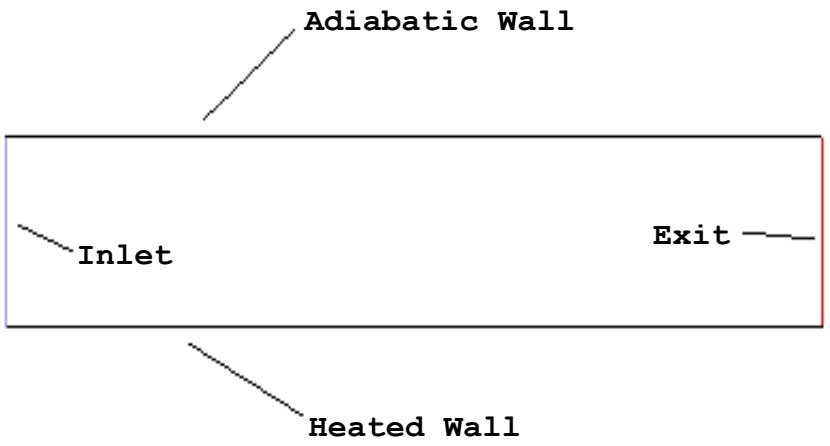


Example 2: Sinusoidal Wall Temperature

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

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

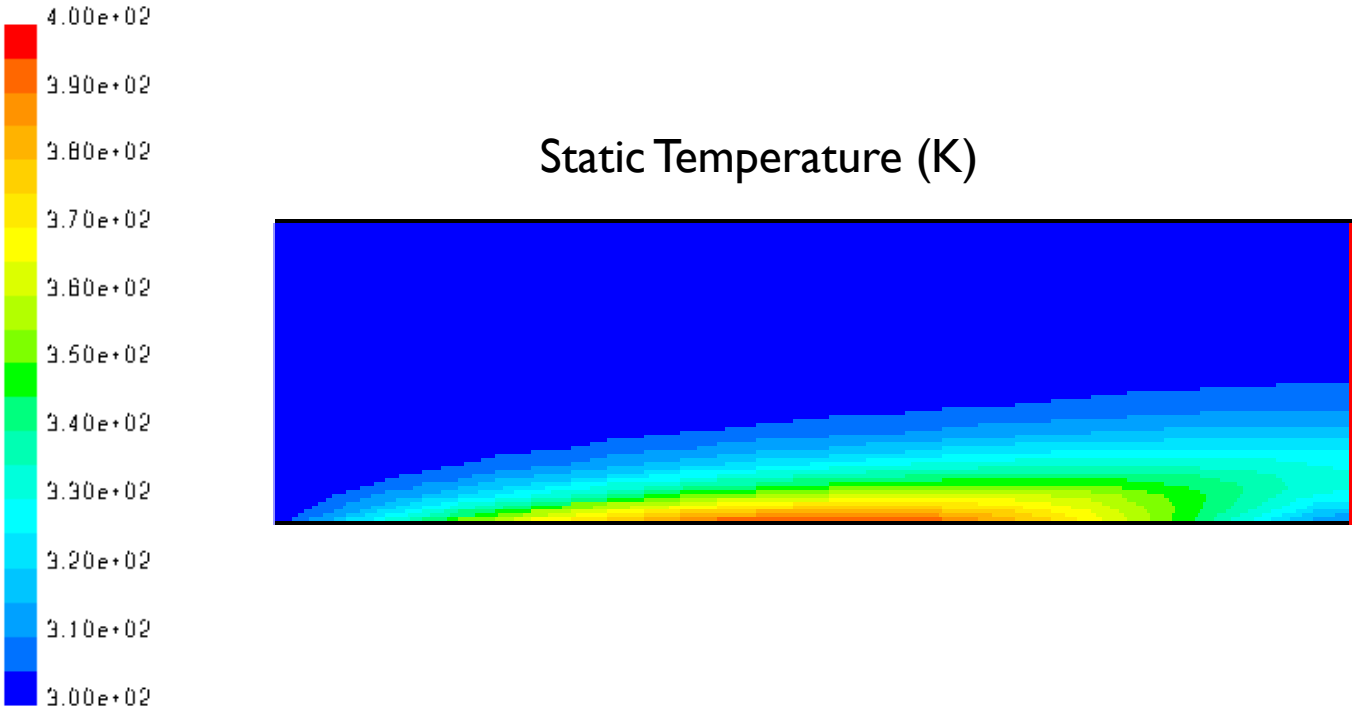
- The boundary condition can be implemented in DEFINE_PROFILE in which a call to F_CENTROID to retrieve the x-coordinates of the cell faces on the heated wall



Temperature	`F_PROFILE(f, t, pos) = 300.+100.*sin(PI*xc[0]/0.005);`

Example 2: Sinusoidal Wall Temperature

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



Source Terms (1)

- Source terms can be added to the governing equations, including
  - continuity
  - momentum
  - k, ε
  - energy
  - species
  - user-defined scalars
- Energy source term UDFs may also be defined for solid zones
- **`DEFINE_SOURCE(name,c,t,dS,eqn)`** is the macro for supplying equation source terms
- 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 (2)

- **`DEFINE_SOURCE(name,c,t,dS,eqn)`** provides the mechanism for adding source terms to the governing equations. (Note: source terms are added to each cell in the continuum zone, hence no loop over cells are needed)
- A source term can be linearized as follows:

$$S_\phi = A + B\phi = \underbrace{\left(S^* - \left(\frac{\partial S_\phi}{\partial \phi}\right)^* \phi^*\right)}_{A} + \underbrace{\left(\frac{\partial S_\phi}{\partial \phi}\right)^*}_{B} \phi$$

where superscript "*" stands for the value based on the previous iteration.

- *B* (dS[eqn]) and *S** (which is returned by the macro) are supplied by the user. The solver will use the linearization automatically if it enhances numerical stability; otherwise *S** is used
- It is hooked up to the cell zone(s) in the boundary-condition panel

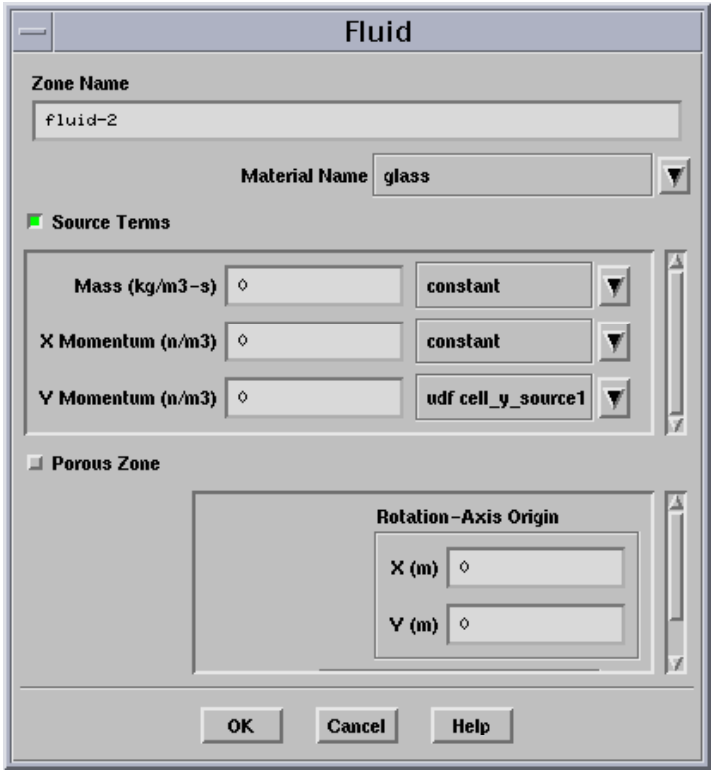
Source Terms (3)

- 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;
}
```

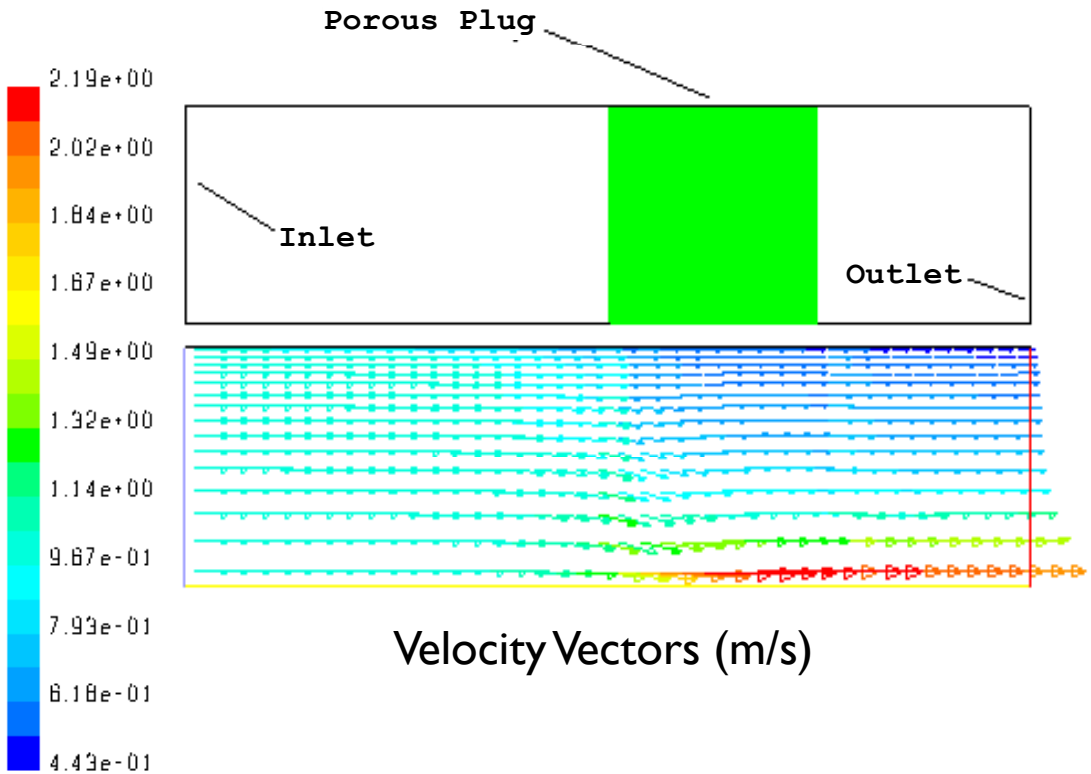
Source Terms (4)

- To activate source terms go to **Define/Boundary Conditions, select fluid-1** and click on **Source Terms**



Example 3: Position Dependent Porous Media

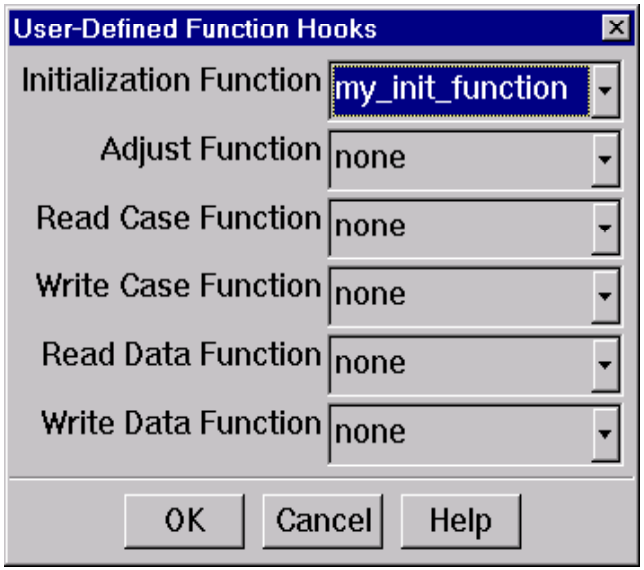
- Channel flow with porous plug
- x-momentum loss is linear in y-position, starting from zero at lower wall
- Fluid flows preferentially near the bottom of the channel



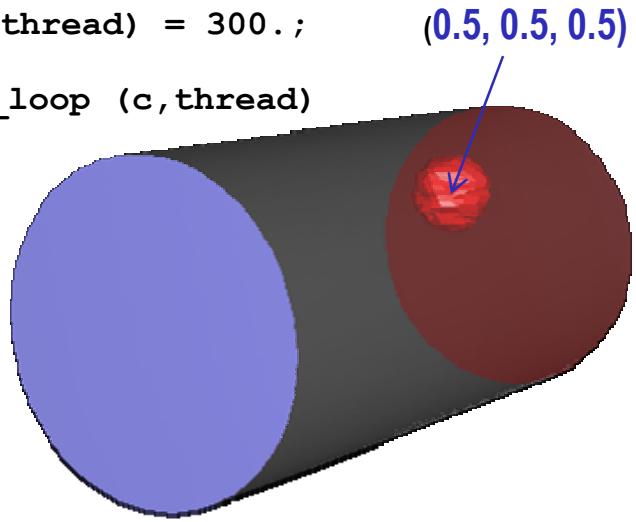
Velocity Vectors (m/s)

Initialization UDF

- DEFINE_INIT initializes solutions for the entire domain (similar to "patching" of values)
- Executed once at the beginning of solution process
- Initialization Function hook appears under **`Define/User_Defined/Function_hooks`**...

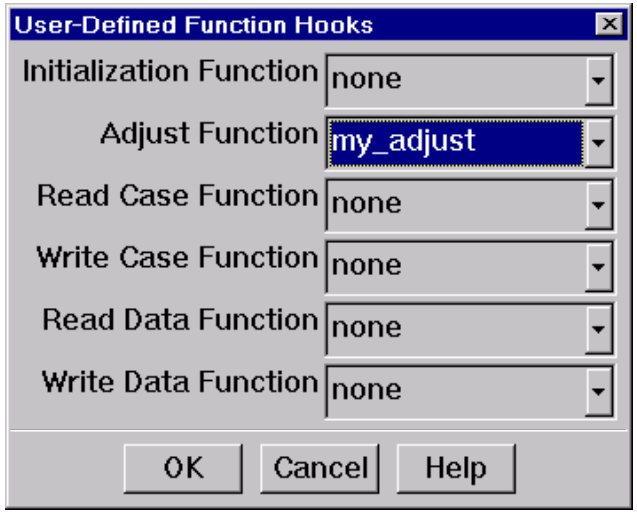



```
#include "udf.h"
DEFINE_INIT(my_init_function,domain)
{
  cell_t c;
  Thread *thread;
  real xc[ND_ND];
  thread_loop_c (thread,domain)
 {
      begin_c_loop (c,thread)
    {
      C_CENTROID(xc,c,thread);
if (sqrt(ND_SUM(pow(xc[0]-0.5,2.),
     pow(xc[1] - 0.5,2.),
    pow(xc[2] - 0.5,2.))) < 0.25)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



Adjust Function

- 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…**

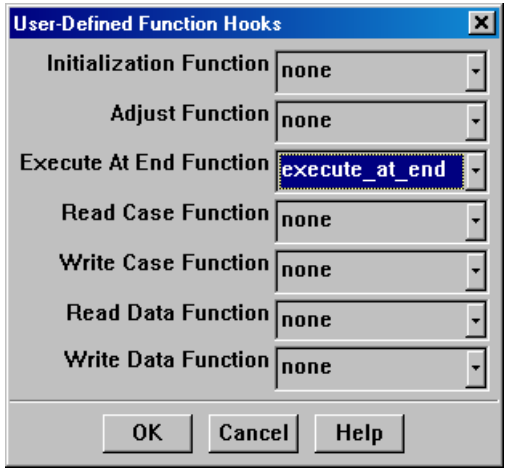



```
DEFINE_ADJUST(my_adjust, domain)
{
/* Integrate dissipation. */
 real sum_diss=0.;
 Thread *t;
 cell_t c;
 thread_loop_c (t,domain)
 {
  begin_c_loop (c,t)
  sum_diss += C_D(c,t)* C_VOLUME(c,t);
  end_c_loop (c,t)
 }
 Message("Volume integral of turbulent
   dissipation : %g\n",sum_diss);
}
```

Execute_at_End Function

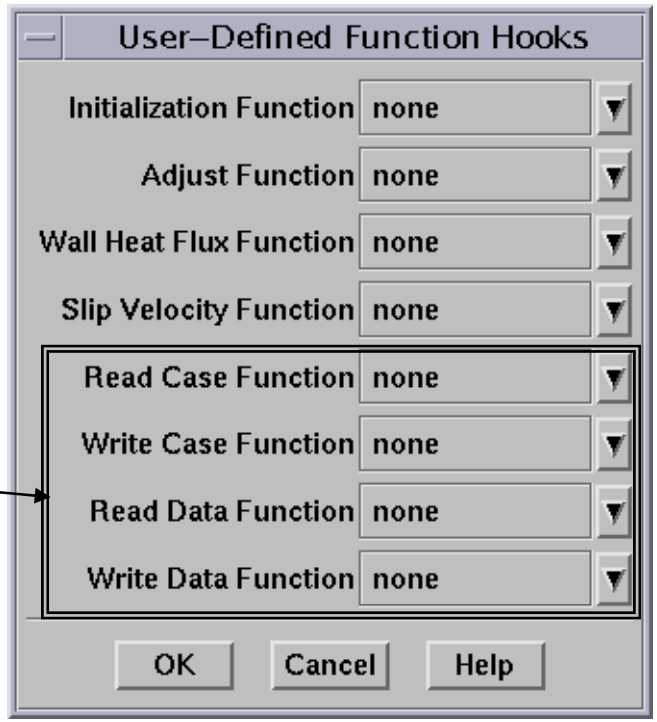
- This is a general purpose macro executed at the end of
  - an iteration in a steady state run, or
  - at the end of a time step in a transient run
- UDF for integrating turbulent dissipation and printing it to console window at the end of the current iteration or time step
- This Function appears under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

```
#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 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
   */
}
```

User-defined Properties

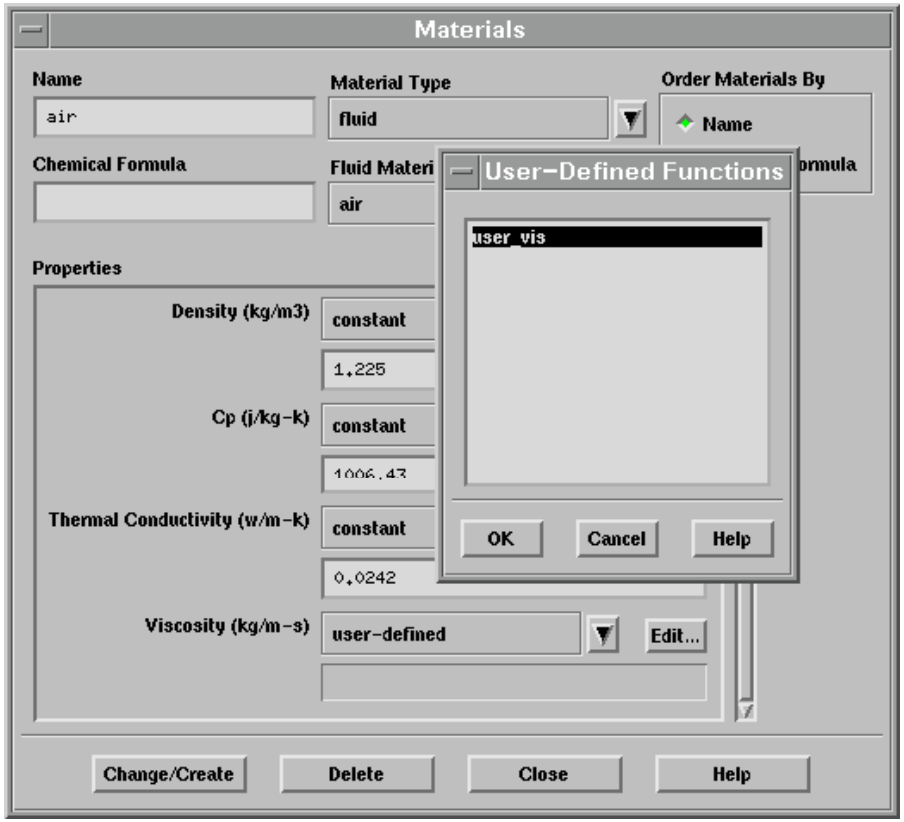
- UDF's can be used to define
  - Viscosity
  - Thermal Conductivity
  - Mass Diffusivity
  - Density
- UDF's cannot be used to define specific heat
- The function is called for every cell in the zone

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

```
#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;
}
```

User-defined Properties (2)

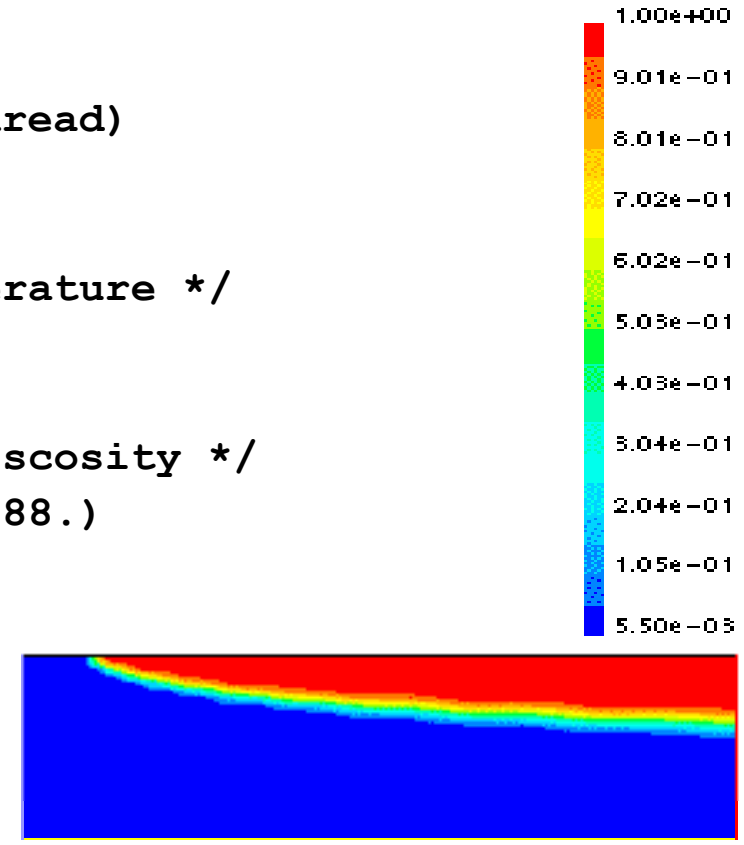
- To activate the UDF, select user-defined from the property drop down list
- When you select the user-defined option, a panel will appear with the names of your UDF's
- Select the name of the appropriate UDF



Example: 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.

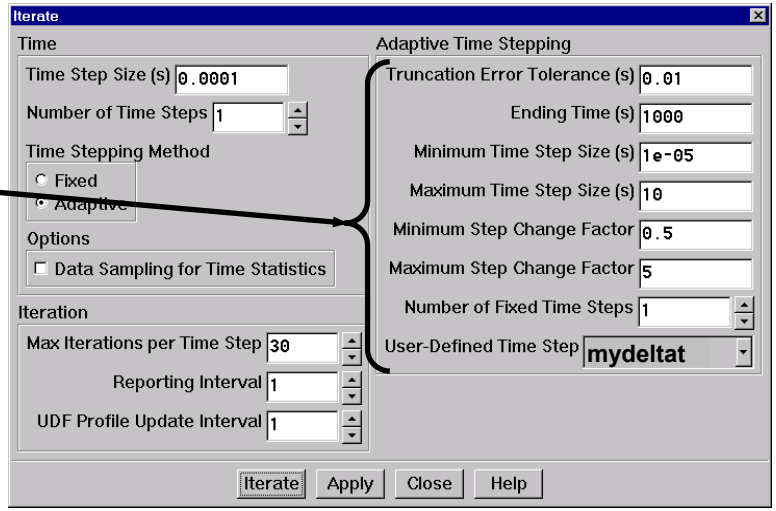
```
#include "udf.h"
DEFINE_PROPERTY(user_vis, cell, thread)
{real temp, mu_lam;
temp = C_T(cell, thread);
{/* Limit viscosity for high temperature */
if (temp > 288.)
  mu_lam = 5.5e-3;
/* Otherwise, use a profile for viscosity */
else if (temp >= 286. && temp <= 288.)
  mu_lam = 143.2135-0.49725*temp;
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 a user control the timestep based on any custom logic/algorithm



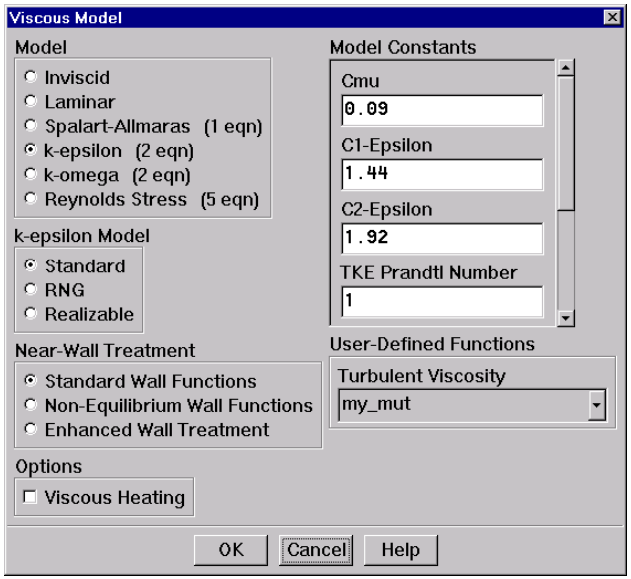

```
#include "udf.h"
DEFINE_DELTAT(mydeltat, domain)
{
real time_step;
real flow_time =
   RP_Get_Real("flow-time");
if (flow_time < 0.5)
   time_step = 0.1;
 else
   time_step = 0.2;
return time_step;
}
```

DEFINE_TURBULENT_VISCOSITY

- 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**

```
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;
}
```

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



Additional Macros

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

```
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