



User Defined Functions

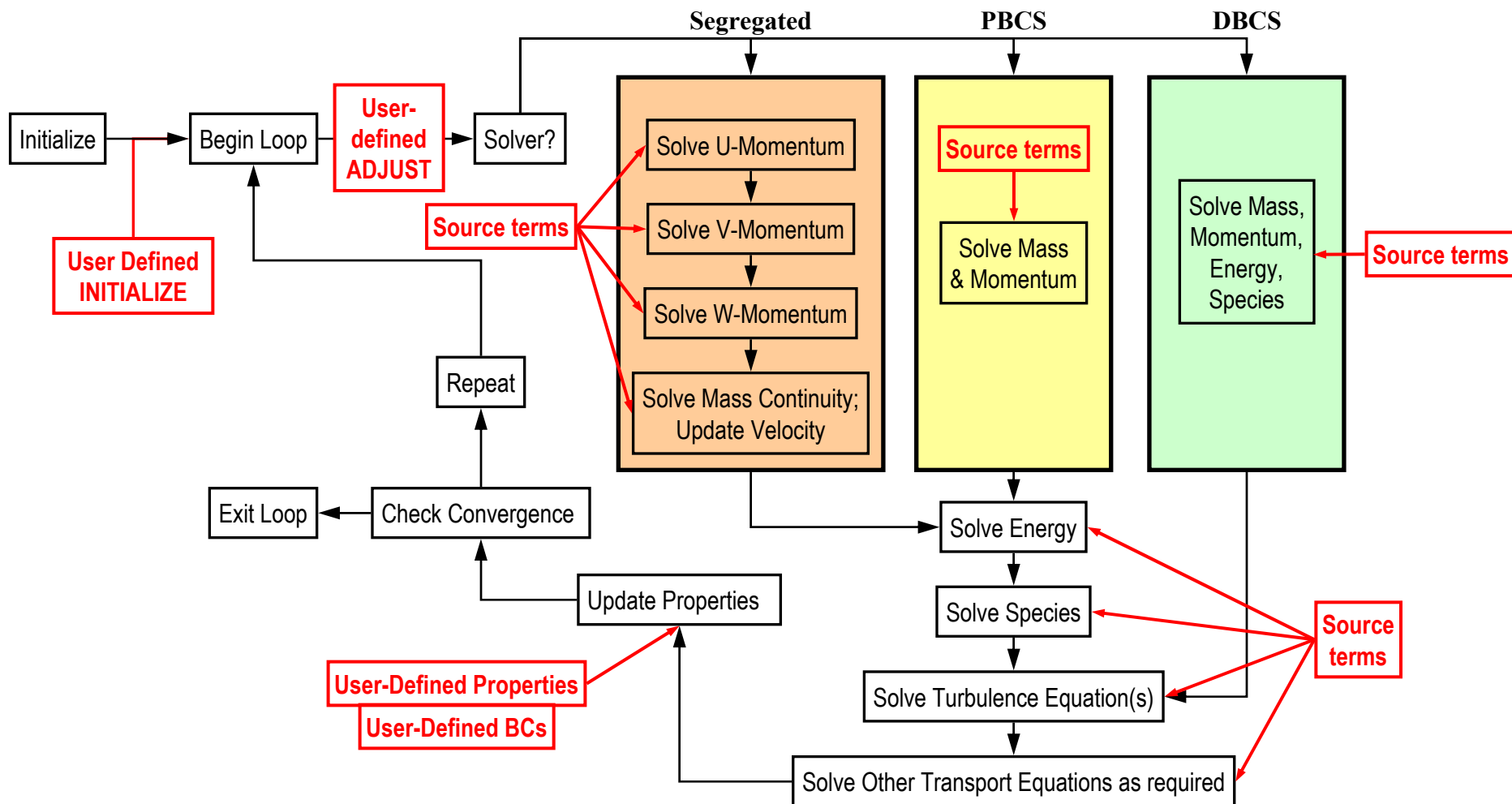
Introductory FLUENT Training



Introduction

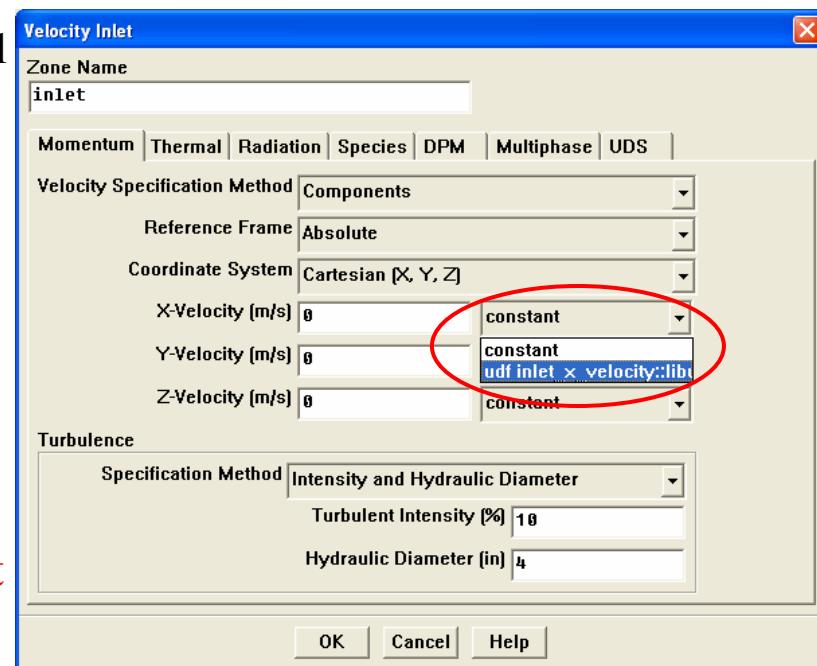
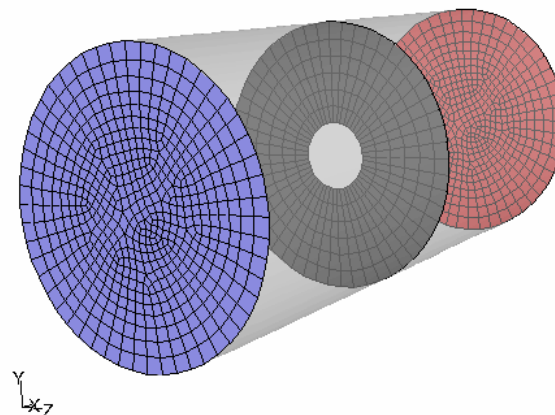
- ◆ What is a User Defined Function?
 - A UDF is a routine (programmed by the user) written in C which can be dynamically linked with the solver.
 - Standard C functions
 - ◆ Trigonometric, exponential, control blocks, do-loops, file i/o, etc.
 - Pre-Defined Macros
 - ◆ Allows access to field variable, material property, and cell geometry data.
- ◆ Why build UDF's?
 - Standard interface cannot be programmed to anticipate all needs.
 - Customization of boundary conditions, source terms, reaction rates, material properties, etc.
 - Adjust functions (once per iteration)
 - Execute on Demand functions
 - Solution Initialization

User Access to the FLUENT Solver



UDF Basics

- ◆ UDF's assigns values (e.g., boundary data, source terms) to individual cells and cell faces in fluid and boundary zones
 - In a UDF, **zones** are referred to as **threads**
 - A looping macro is used to access individual cells belonging to a thread.
 - For example, a face-loop macro visits 563 faces on face zone 3 (named inlet).
 - ◆ Position of each face is available to calculate and assign spatially varying properties
 - Thread and variable references are automatically passed to the UDF when assigned to a boundary zone in the GUI.
- ◆ Values returned to the solver by UDFs must be in SI units.



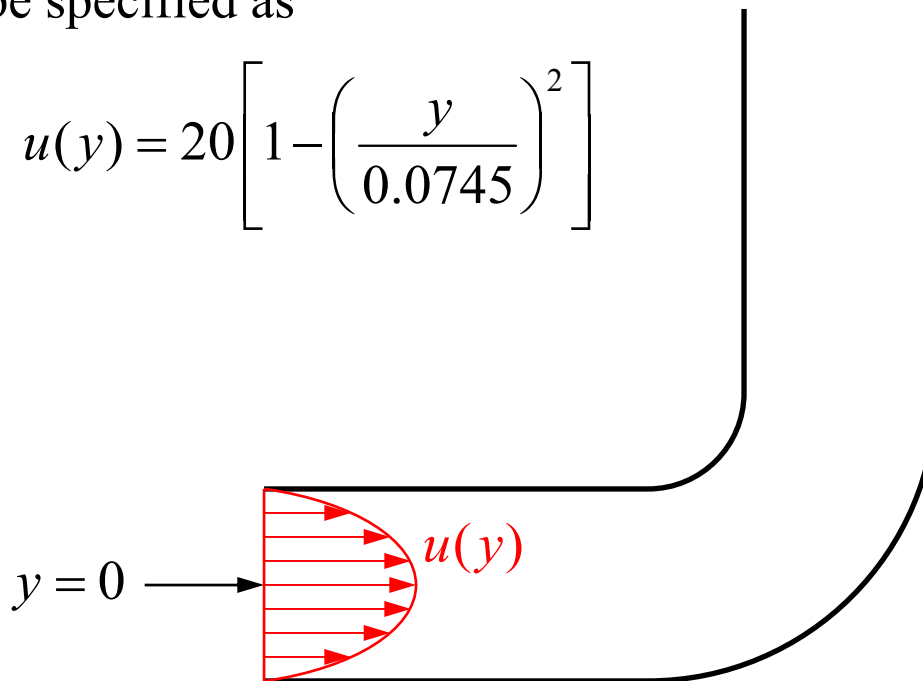
Using UDFs in the Solvers

- ◆ The basic steps for using UDFs in FLUENT are as follows:
 1. Create a file containing the UDF source code
 2. Start the solver and read in your case/data files
 3. Interpret or Compile the UDF
 4. Assign the UDF to the appropriate variable and zone in BC panel.
 5. Set the UDF update frequency in the Iterate panel
 6. Run the calculation

Example – Parabolic Inlet Velocity Profile

- ◆ We would like to impose a parabolic inlet velocity to the 2D elbow shown.
- ◆ The x velocity is to be specified as

$$u(y) = 20 \left[1 - \left(\frac{y}{0.0745} \right)^2 \right]$$



Step 1 – Prepare the Source Code

- ◆ The **DEFINE_PROFILE** macro allows the function **inlet_x_velocity** to be defined.
 - All UDFs begin with a **DEFINE_** macro.
 - **inlet_x_velocity** will be identifiable in solver GUI.
 - **thread** and **nv** are arguments of the **DEFINE_PROFILE** macro, which are used to identify the zone and variable being defined, respectively.
 - The macro **begin_f_loop** loops over all faces **f**, pointed by thread
- ◆ The **F_CENTROID** macro assigns cell position vector to **x[]**
- ◆ The **F_PROFILE** macro applies the velocity component to face **f**

```
#include "udf.h"
DEFINE_PROFILE(inlet_x_velocity, thread, nv)
{
    float x[3]; /* Position vector*/
    float y;
    face_t f;
    begin_f_loop(f, thread)
    {
        F_CENTROID(x, f, thread);
        y = x[1];
        F_PROFILE(f, thread, nv)
            = 20.*(1.- y*y/(.0745*.0745));
    }
    end_f_loop(f, thread)
}
```

Step 3 – Interpret or Compile the UDF

◆ Interpreted UDF

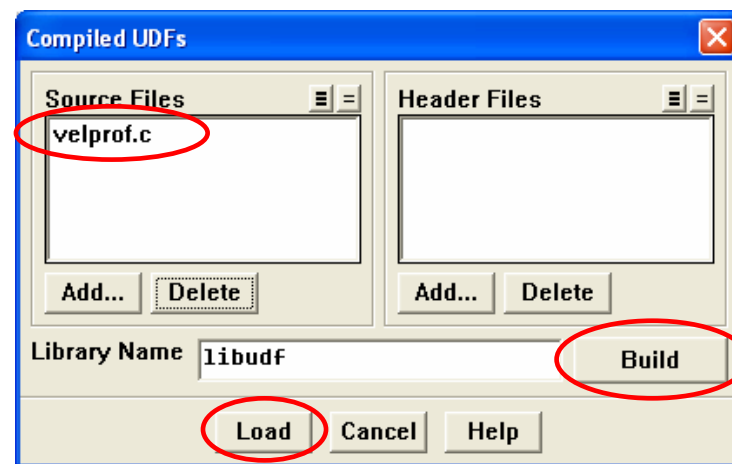
Define → User-Defined → Functions → Interpreted...



- ◆ Add the UDF source code to the Source File Name list.
- ◆ Click Interpret.
- ◆ The assembly language code will display in the FLUENT console.

◆ Compiled UDF

Define → User-Defined → Functions → Compiled...



- ◆ Add the UDF source code to the Source Files list.
- ◆ Click Build to create UDF library.
- ◆ Click Load to load the library.
- ◆ You can also unload a library if needed.

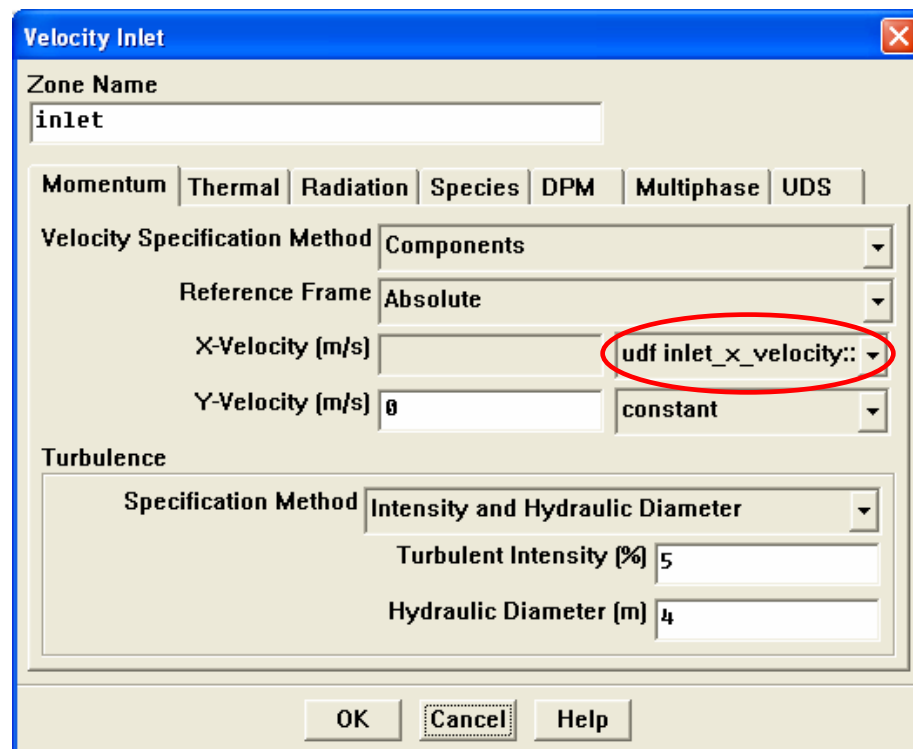
Define → User-Defined → Functions → Manage...

Interpreted vs. Compiled UDFs

- ◆ Functions can either be read and **interpreted** at run time or **compiled** and grouped into a shared library that is linked with the standard FLUENT executable.
- ◆ Interpreted code vs. compiled code
 - Interpreted
 - Interpreter is a large program that sits in the computer's memory.
 - Executes code on a "line by line" basis instantaneously
 - Advantage – Does not require a third-party compiler.
 - Disadvantage – Interpreter is slow and takes up memory.
 - Compiled (refer to the FLUENT User's Guide for instructions)
 - UDF code is translated once into machine language (object modules).
 - Efficient way to run UDFs.
 - Creates shared libraries which are linked with the rest of the solver
 - Overcomes many interpreter limitations such as mixed mode arithmetic, structure references, etc.

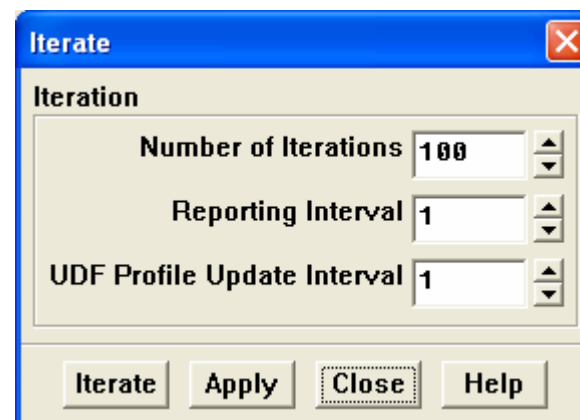
Step 4 – Activate the UDF

- ◆ Open the boundary condition panel for the surface to which you would like to apply the UDF.
- ◆ Switch from Constant to the UDF function in the X-Velocity drop-down list.



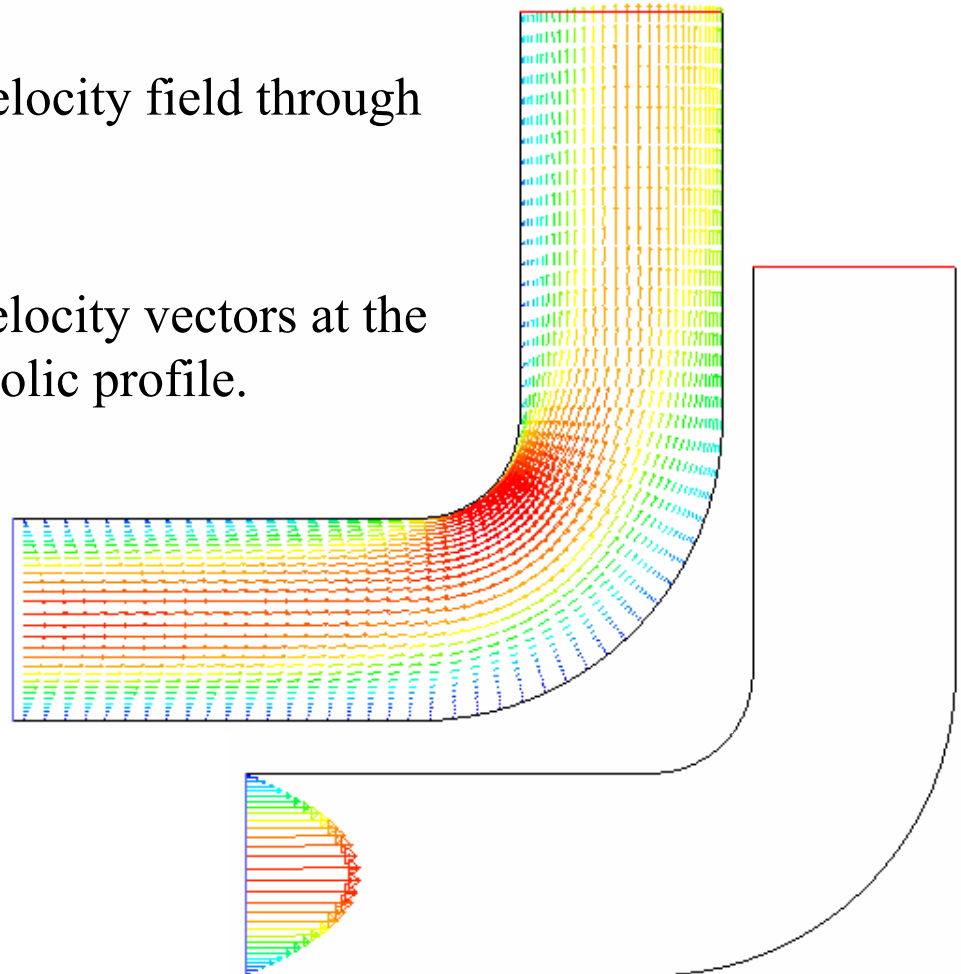
Steps 5 and 6 – Run the Calculations

- ◆ You can change the UDF Profile Update Interval in the Iterate panel (default value is 1).
 - This setting controls how often (either iterations or time steps if unsteady) the UDF profile is updated.
- ◆ Run the calculation as usual.



Numerical Solution of the Example

- ◆ The figure at right shows the velocity field through the 2D elbow.
- ◆ The bottom figure shows the velocity vectors at the inlet. Notice the imposed parabolic profile.



Macros

- ◆ Macros are functions defined by FLUENT.
 - **DEFINE_** macros allows definitions of UDF functionality.
 - Variable access macros allow access to field variables and cell information.
 - Utility macros provide looping capabilities, thread identification, vector and numerous other functions.
- ◆ Macros are defined in header files.
 - The **udf.h** header file must be included in your source code.
#include "udf.h"
 - The header files must be accessible in your path.
 - Typically stored in **Fluent.Inc/src/** directory.
- ◆ A list of often used macros is provided in the UDF User's Guide.

Help → More Documentation...

DEFINE Macros

- ◆ Any UDF you write must begin with a **DEFINE_** macro:
 - 18 ‘general purpose’ macros and 13 DPM and multiphase related macros (not listed):

DEFINE_ADJUST(name, domain) ; general purpose UDF called every iteration
DEFINE_INIT(name, domain) ; UDF used to initialize field variables
DEFINE_ON_DEMAND(name) ; defines an ‘execute-on-demand’ function
DEFINE_RW_FILE(name, fp) ; customize reads/writes to case/data files
DEFINE_PROFILE(name, thread, index) ; defines boundary profiles
DEFINE_SOURCE(name, cell, thread, dS, index) ; defines source terms
DEFINE_HEAT_FLUX(name, face, thread, c0, t0, cid, cir) ; defines heat flux
DEFINE_PROPERTY(name, cell, thread) ; defines material properties
DEFINE_DIFFUSIVITY(name, cell, thread, index) ; defines UDS and species diffusivities
DEFINE_UDS_FLUX(name, face, thread, index) ; defines UDS flux terms
DEFINE_UDS_UNSTEADY(name, cell, thread, index, apu, su) ; defines UDS transient terms
DEFINE_SR_RATE(name, face, thread, r, mw, yi, rr) ; defines surface reaction rates
DEFINE_VR_RATE(name, cell, thread, r, mw, yi, rr, rr_t) ; defines vol. reaction rates
DEFINE_SCAT_PHASE_FUNC(name, cell, face) ; defines scattering phase function for DOM
DEFINE_DELTAT(name, domain) ; defines variable time step size for unsteady problems
DEFINE_TURBULENT_VISCOSITY(name, cell, thread) ; defines procedure for calculating turbulent viscosity
DEFINE_TURB_PREMIX_SOURCE(name, cell, thread, turbflamespeed, source) ; defines turb. flame speed
DEFINE_NOX_RATE(name, cell, thread, nox) ; defines NOx production and destruction rates

Thread and Looping Utility Macros

<code>cell_t c;</code>	Defines c as a cell thread index	} <code>cell_t, face_t, Thread, Domain</code> are part of FLUENT UDF data structure
<code>face_t f;</code>	Defines f as a face thread index	
<code>Thread *t; t</code>	is a pointer to a thread	
<code>Domain *d; d</code>	is a pointer to collection of all threads	

`thread_loop_c(t, d){}` Loop that visits all cell threads `t` in domain `d`

`thread_loop_f(t, d){}` Loop that visits all face threads `t` in domain `d`

`begin_c_loop(c, ct) {} end_c_loop(c, ct)`
Loop that visits all cells `c` in cell thread `ct`

`begin_f_loop(f, ft) {} end_f_loop(f, ft)`
Loop that visits all faces `f` in a face thread `ft`

`c_face_loop(c, t, n){}` Loop that visits all faces of cell `c` in thread `t`

`Thread *tf = Lookup_Thread(domain, ID);` Returns the thread pointer of zone `ID`

`ID = THREAD_ID(tf);` Returns the zone integer ID of thread pointer `tf`

Geometry and Time Macros

<code>C_NNODES (c, t) ;</code>	Returns nodes/cell
<code>C_NFACES (c, t) ;</code>	Returns faces/cell
<code>F_NNODES (f, t) ;</code>	Returns nodes/face
<code>C_CENTROID (x, c, t) ;</code>	Returns coordinates of cell centroid in array <code>x[]</code>
<code>F_CENTROID (x, f, t) ;</code>	Returns coordinates of face centroid in array <code>x[]</code>
<code>F_AREA (A, f, t) ;</code>	Returns area vector in array <code>A[]</code>
<code>C_VOLUME (c, t) ;</code>	Returns cell volume
<code>C_VOLUME_2D (c, t) ;</code>	Returns cell volume (axisymmetric domain)
<code>real flow_time() ;</code>	Returns actual time
<code>int time_step ;</code>	Returns time step number
<code>RP_Get_Real("physical-time-step") ;</code>	Returns time step size

Cell Field Variable Macros

$C_R(c, t)$; Density
 $C_P(c, t)$; Pressure
 $C_U(c, t)$; U-velocity
 $C_V(c, t)$; V-velocity
 $C_W(c, t)$; W-velocity
 $C_T(c, t)$; Temperature
 $C_H(c, t)$; Enthalpy
 $C_K(c, t)$; Turbulent kinetic energy (k)
 $C_D(c, t)$; Turbulent dissipation rate (ϵ)
 $C_O(c, t)$; Specific dissipation of TKE (ω)
 $C_YI(c, t, i)$; Species mass fraction
 $C_UDSI(c, t, i)$; UDS scalars
 $C_UDMI(c, t, i)$; UDM scalars

$C_DUDX(c, t)$; Velocity derivative
 $C_DUDY(c, t)$; Velocity derivative
 $C_DUDZ(c, t)$; Velocity derivative

$C_DVDX(c, t)$; Velocity derivative
 $C_DVDY(c, t)$; Velocity derivative
 $C_DVDZ(c, t)$; Velocity derivative
 $C_DWDX(c, t)$; Velocity derivative
 $C_DWDY(c, t)$; Velocity derivative
 $C_DWDZ(c, t)$; Velocity derivative

$C_MU_L(c, t)$; Laminar viscosity
 $C_MU_T(c, t)$; Turbulent viscosity
 $C_MU_EFF(c, t)$; Effective viscosity
 $C_K_L(c, t)$; Laminar thermal conductivity
 $C_K_T(c, t)$; Turbulent thermal conductivity
 $C_K_EFF(c, t)$; Effective thermal conductivity
 $C_CP(c, t)$; Specific heat
 $C_RGAS(c, t)$; Gas constant
 $C_DIFF_L(c, t)$; Laminar species diffusivity
 $C_DIFF_EFF(c, t, i)$; Effective species diffusivity

Face Field Variable Macros

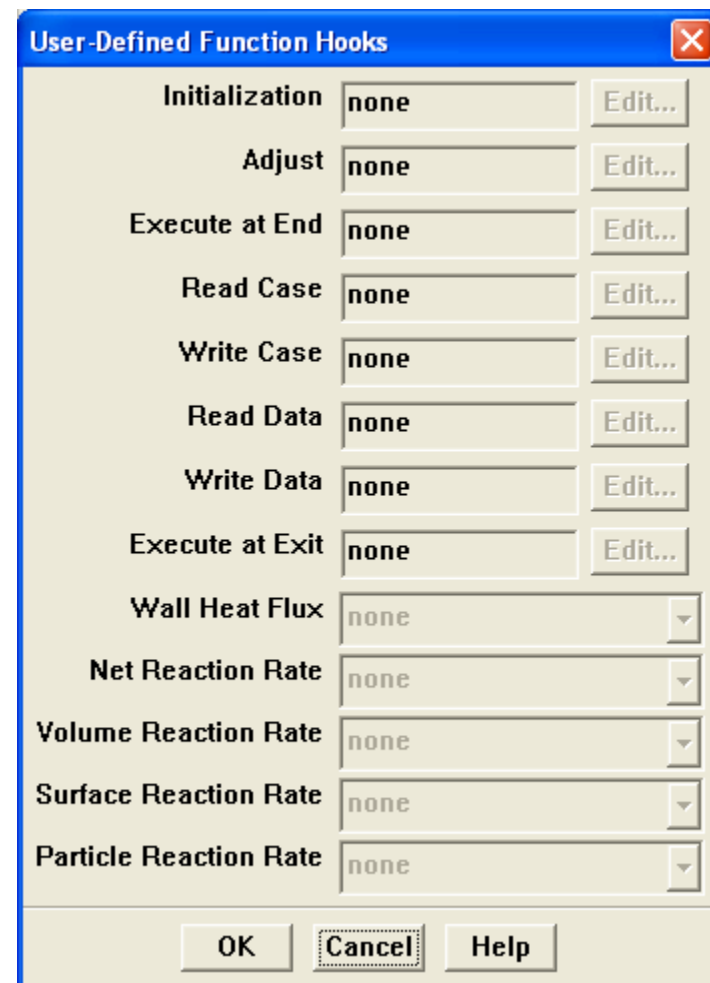
- ◆ Face field variables are only available when using the segregated solver and generally, only at exterior boundaries.

<code>F_R(f,t) ;</code>	Density
<code>F_P(f,t) ;</code>	Pressure
<code>F_U(f,t) ;</code>	U-velocity
<code>F_V(f,t) ;</code>	V-velocity
<code>F_W(f,t) ;</code>	W-velocity
<code>F_T(f,t) ;</code>	Temperature
<code>F_H(f,t) ;</code>	Enthalpy
<code>F_K(f,t) ;</code>	Turbulent KE
<code>F_D(f,t) ;</code>	TKE dissipation
<code>F_O(f,t) ;</code>	Specific dissipation of tke
<code>F_YI(f,t,i) ;</code>	Species mass fraction
<code>F_UDSI(f,t,i) ;</code>	UDS scalars
<code>F_UDMI(f,t,i) ;</code>	UDM scalars
<code>F_FLUX(f,t) ;</code>	Mass flux across face <code>f</code> , defined out of domain at boundaries.

Other UDF Applications

- ◆ In addition to defining boundary values, source terms, and material properties, UDFs can be used for:

- Initialization
 - Executes once per initialization.
- Solution adjustment
 - Executes every iteration.
- Wall heat flux
 - Defines fluid-side diffusive and radiative wall heat fluxes in terms of heat transfer coefficients
 - Applies to all walls
- User-defined surface and volumetric reactions
- Read/write to/from case and data files
 - Read order and write order must be same.
- Execute-on-Demand capability
 - Does not participate in the solver iterations

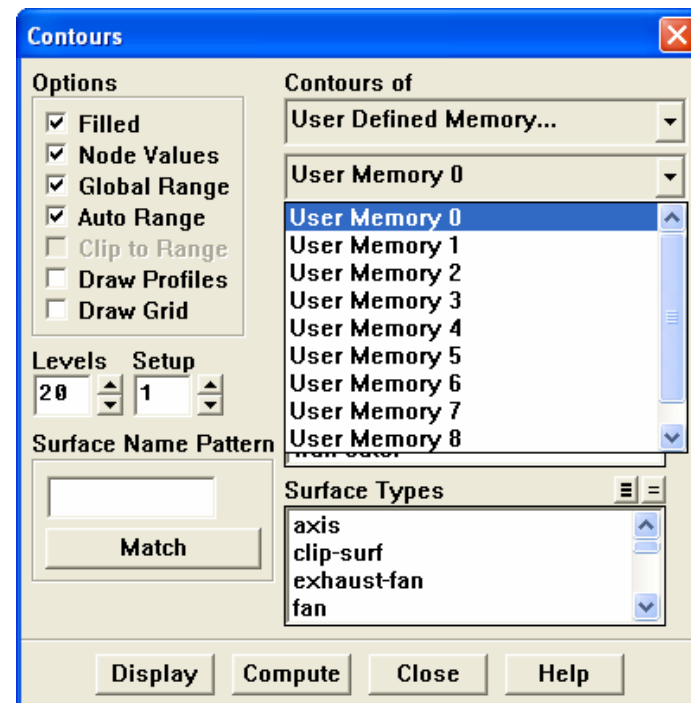


User Defined Memory

◆ User-allocated memory

Define → User-Defined → Memory...

- Up to 500 field variables can be defined.
- Can be accessed by UDFs:
 - `C_UDMI (cell, thread, index) ;`
 - `F_UDMI (face, thread, index) ;`
- Can be accessed for post-processing.
- Information is stored in data file.



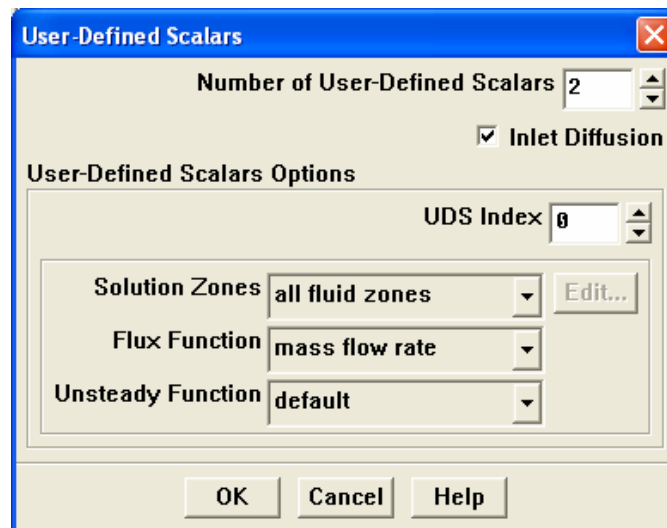
User Defined Scalars

- ◆ FLUENT can solve up to 50 generic transport equations for user-defined scalars.

$$\frac{\partial \phi_k}{\partial t} + \frac{\partial}{\partial x_i} \left(F_i \phi_k - \Gamma_k \frac{\partial \phi_k}{\partial x_i} \right) = S_{\phi_k} \quad k = 1, 2, \dots, N_{\text{scalar}}$$

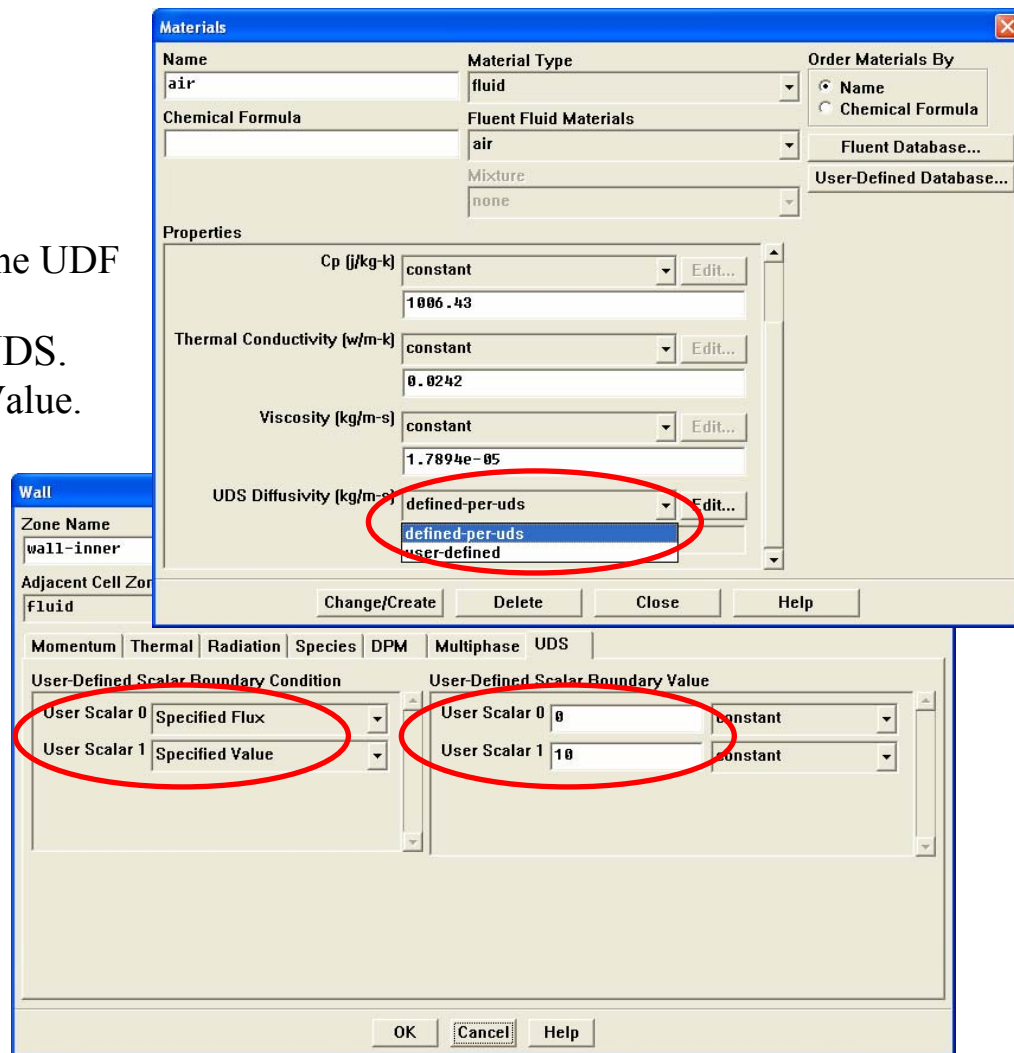
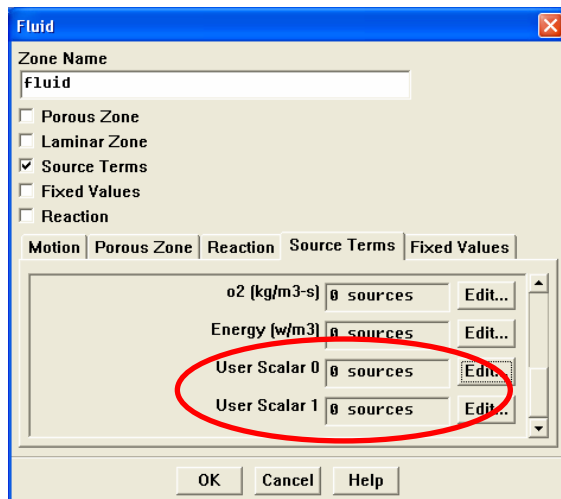
Define → User-Defined → Scalars...

- Number of UDS variables
 - Zone(s) on which to calculate UDS transport
 - Flux Function, F
 - **DEFINE_UDS_FLUX**(name, face, thread, index)
 - **DEFINE_UDS_UNSTEADY**(name, cell, thread, index, apu, su)
 - **If** statements are required in order to associate multiple flux and transient functions with each UDS.
- ◆ Example
 - Can be used to determine magnetic and/or electric field in a fluid zone.



User Defined Scalars

- ◆ User must also specify:
 - Source terms, S_ϕ
 - Diffusivity, Γ_ϕ
 - **I**f statements needed to define UDF diffusivities for each UDS.
 - Boundary Conditions for each UDS.
 - Specified Flux or Specified Value.
- ◆ Define as constant or with UDF.



UDF Technical Support

- ◆ Because UDFs can be very complicated, ANSYS does not assume responsibility for the accuracy or stability of solutions obtained using UDFs that are user-generated.
 - Support will be limited to guidance related to communication between a UDF and the FLUENT solver.
 - Other aspects of the UDF development process that include conceptual function design, implementation (writing C code), compilation and debugging of C source code, execution of the UDF, and function design verification will remain the responsibility of the UDF author.
 - A consulting option is available for complex projects.