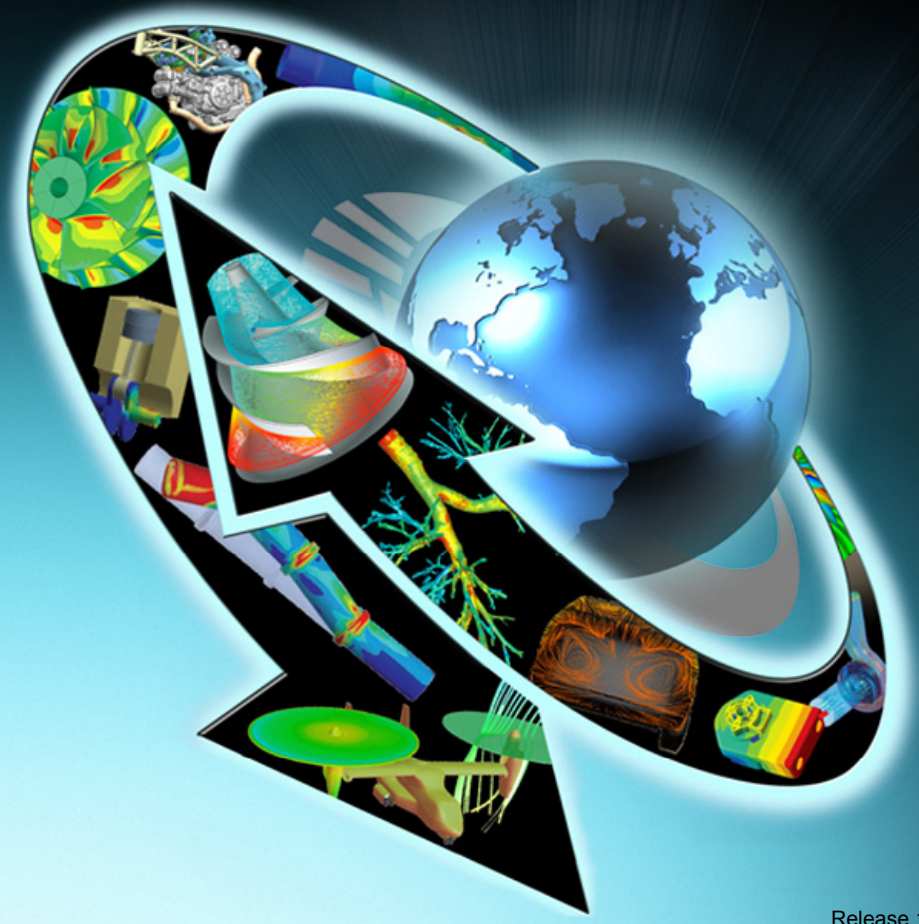
ANSYS®

Customer Training Material

Lecture 8

User Defined Functions (UDFs)



Introduction to ANSYS FLUENT

Outline

- A brief introduction to FLUENT user-defined functions
- Overview of FLUENT data structure and macros
- Two examples
- Where to get more information and help
- UDF support

Introduction

- What is a User Defined Function?
  - A UDF is a function (programmed by the user) written in C which can be dynamically linked with the FLUENT 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 and many utilities

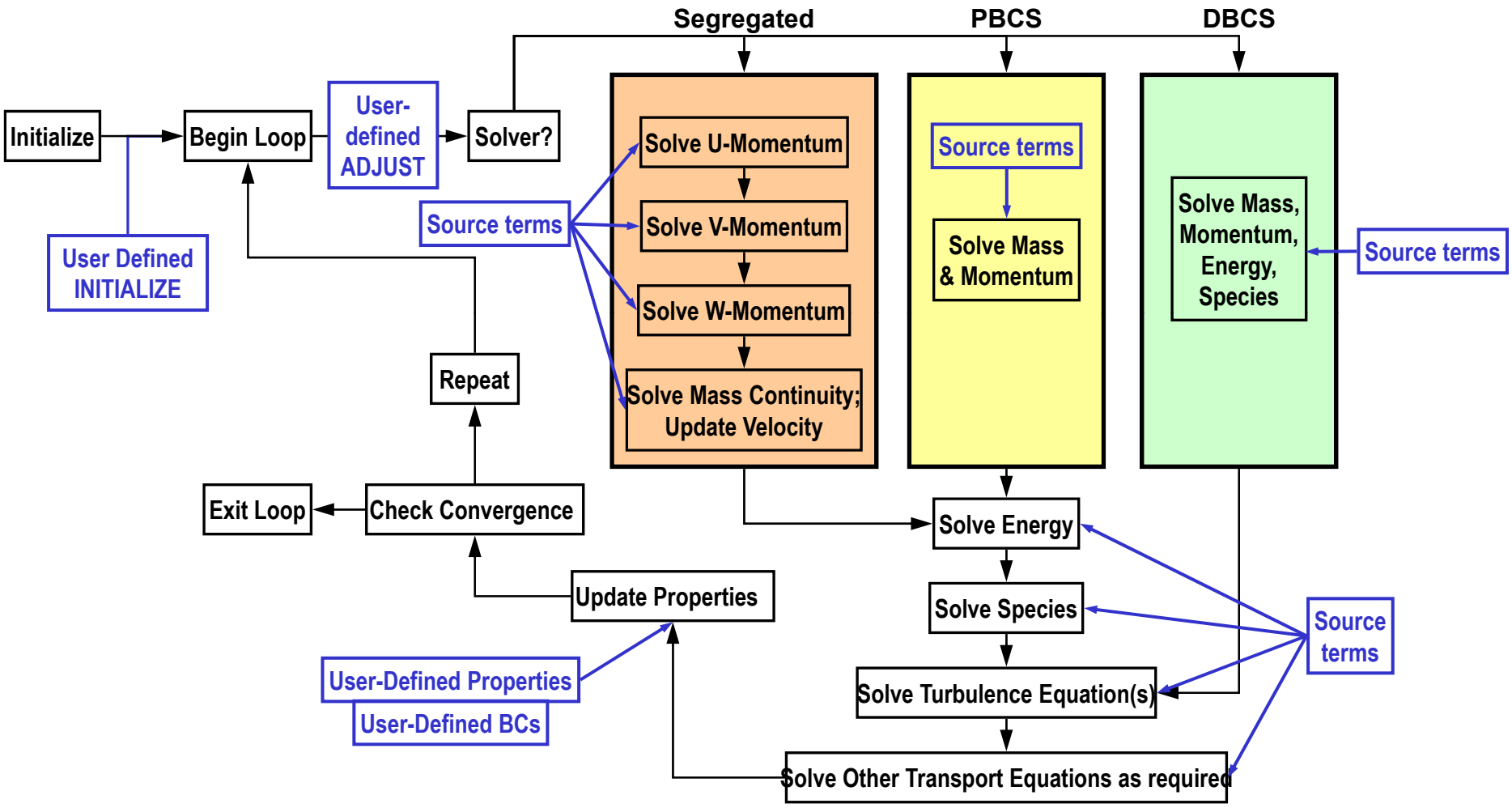
- Why program UDFs?
  - Standard interface cannot be programmed to anticipate all needs:
    - Customization of boundary conditions, source terms, reaction rates, material properties, etc.
    - Customization of physical models
    - User-supplied model equations
    - Adjust functions (once per iteration)
    - Execute on Demand functions
    - Solution Initialization

Interpreted vs. Compiled UDFs

- UDFs can either be run compiled or interpreted.
  - The supported compiler for FLUENT is Microsoft Visual Studio (Standard or Express Editions)

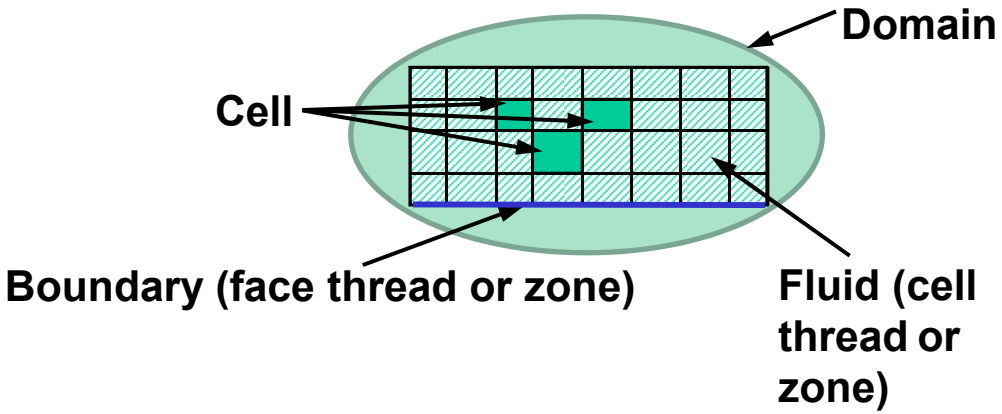
- Interpreted code vs. compiled code
  - Interpreted
    - C++ Interpreter bundled with FLUENT
    - Interpreter executes code on a “line by line” basis instantaneously.
    - Advantage – Does not require a third-party compiler.
    - Disadvantage – Interpreter is slow, and cannot do some functions.
  - Compiled
    - 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.
    - Does require a compilation step between creating/editing your UDF and using it.

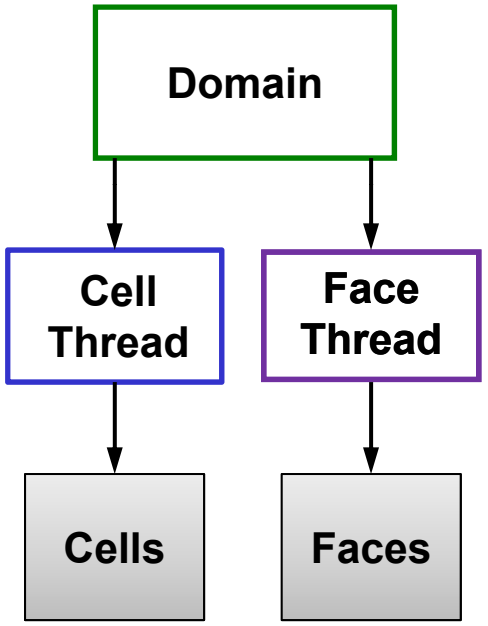
User Access to the FLUENT Solver



Fluent UDF Data Structure (1)

- The cell zones and face zones of a model (in the finite-volume scheme) are accessed in UDFs as `Thread` data types
- `Thread` is a FLUENT-defined data type



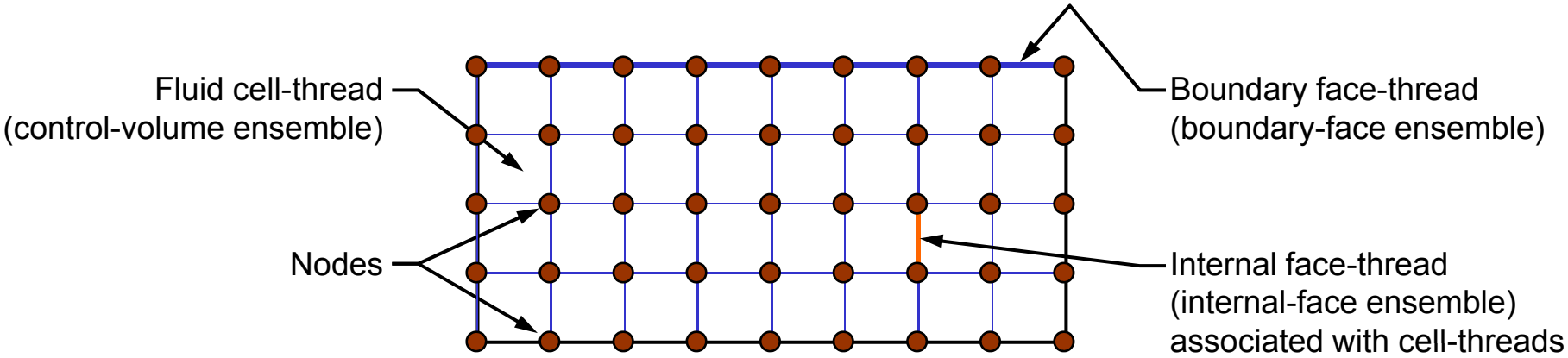


- In order to access data in a thread (zone), we need to provide the correct thread pointer, and use FLUENT provided looping macros to access each member (cell or face) in that thread.

Fluent UDF Data Structure (2)

- `cell_t` declares an integer data type used to identify cells
- `face_t` declares an integer data type used to identify faces

Type	Variable	Meaning of the declaration
`Domain`	`*d;`	`d` is a pointer to domain thread
`Thread`	`*t;`	`t` is a pointer to thread
`cell_t`	`c;`	`c` is cell thread variable
`face_t`	`f;`	`f` is a face thread variable
`Node`	`*node;`	`node` is a pointer to a node.



Loop Macros in UDF

- Several frequently used loop macros:
  - Loop over all cell threads in domain `d`:

```
thread_loop_c(ct,d) {  }
```

  - Loop over face threads in domain `d`:

```
thread_loop_f(ft,d) {  }
```

  - Loop over all cells in a cell thread `t`:

```
begin_c_loop(c, t)
    {…}
end_c_loop (c,t)
```

  - Loop over faces in a face thread `f_thread`:

```
begin_f_loop(f, f_thread)
{ … }
end_f_loop(f, f_thread)
```

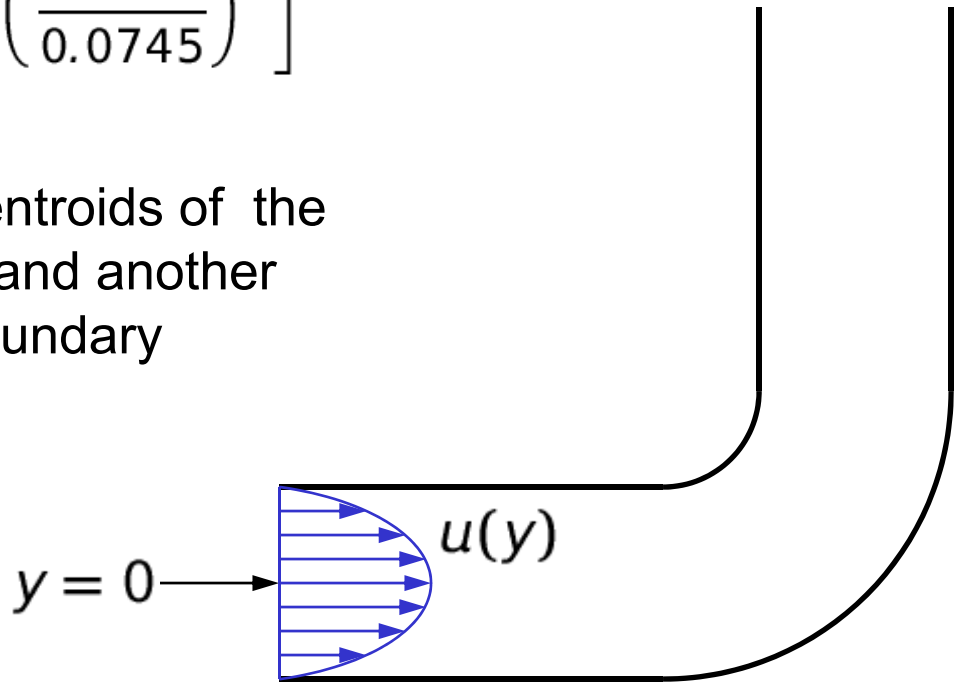
`d`: a domain pointer
`ct, t`: a cell thread pointer
`ft,f_thread`: a face thread pointer
`c`: a cell thread variable
`f`: a face thread variable

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

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

- We need to know the centroids of the inlet faces via a macro, and another macro to perform the boundary condition assignment.



Step 1 – Prepare the Source Code

- The **DEFINE_PROFILE** macro allows the function **x_velocity** to be defined.
  - All UDFs begin with a **DEFINE_** macro
  - **x_velocity** will appear in the 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**
- The code is stored as a text file **inlet_bc.c**

Header file "udf.h" must be included at the top of the program by the **#include** command

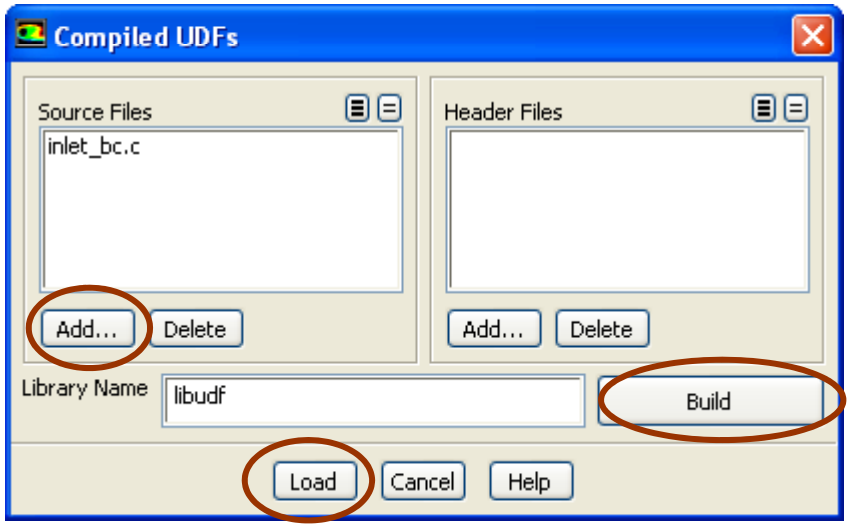
```
#include "udf.h"
DEFINE_PROFILE(x_velocity,thread,nv)
{
  float x[3]; /* an array for the
                 coordinates */
  float y;
  face_t f;   /* f is a face
                 thread index */

  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 2 – Interpret or Compile the UDF

- **Compiled UDF**

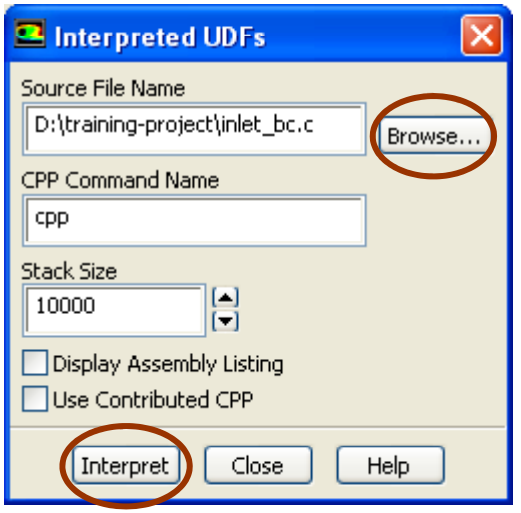
Define → User-Defined → Functions → Compiled



- Add the UDF source code to the Source Files list
- Click Build to compile and link the code
- If no errors, click Load to load the library
- You can also unload a library if needed.
  `/define/user-defined/functions/manage`

- **Interpreted UDF**

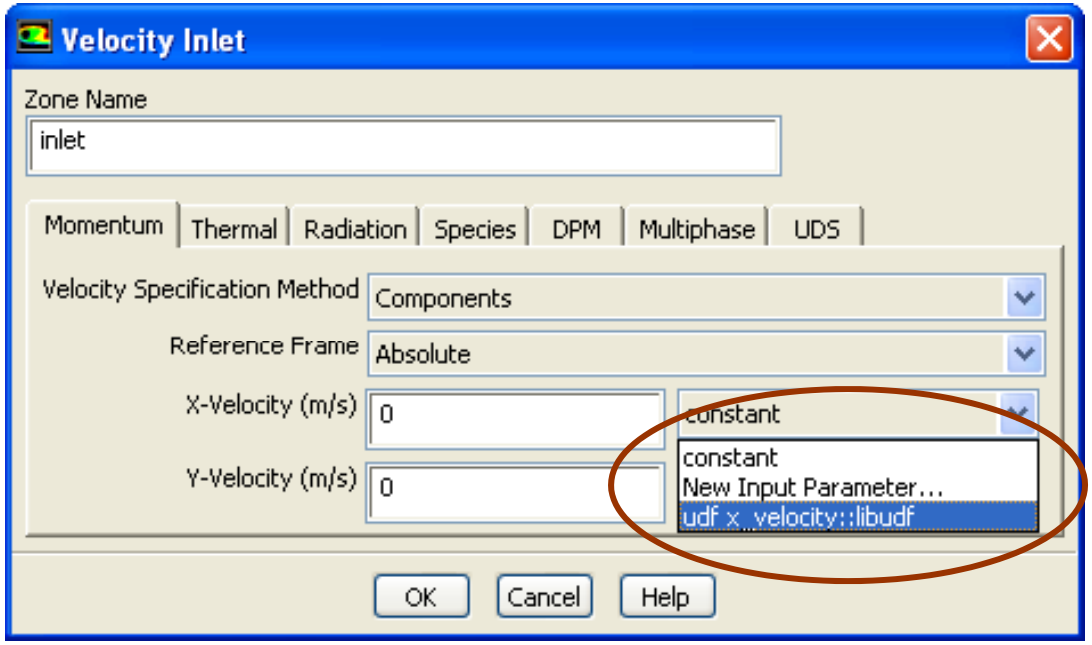
Define → User-Defined → Functions → Interpreted



- Add the UDF source code to the Source File Name list.
- Click Interpret
- The assembly language code will display in the FLUENT console
- Click Close if there is no error

Step 3 – Hook the UDF in FLUENT GUI

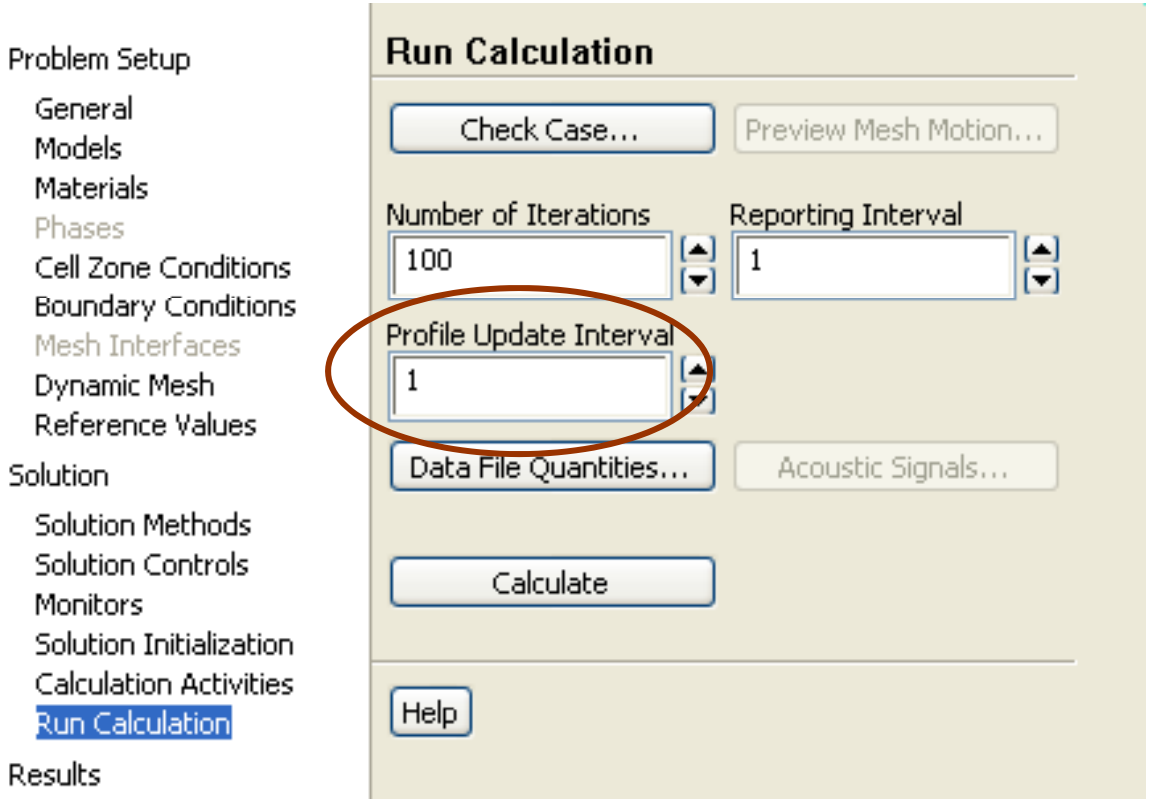
- Open the boundary condition panel for the surface to which you would like to apply the UDF
- Switch from Constant to **`udf x_velocity`** in the drop-down list.
- The macro name is the first argument of **`DEFINE_PROFILE`** in the UDF code



Step 4 – Run the Calculations

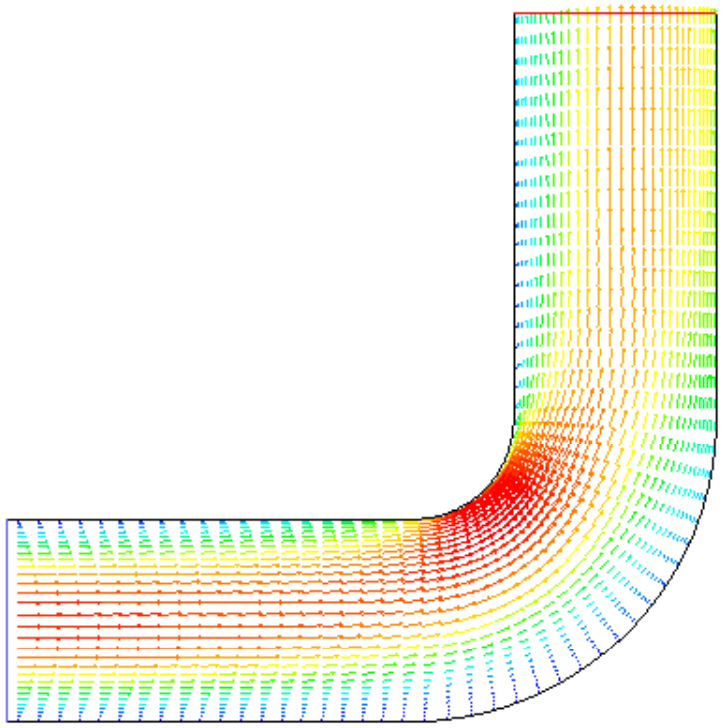
- You can change the Profile Update Interval in the Run Calculation panel (default value is 1).
  - This setting controls how often (either iterations or time steps if unsteady) the UDF profile is updated.

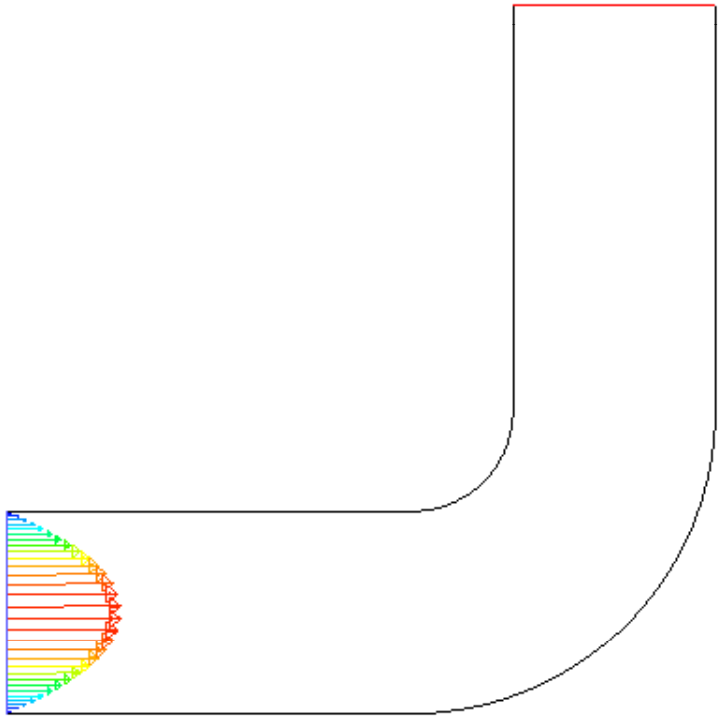
- Run the calculation as usual.



Numerical Solution of the Example

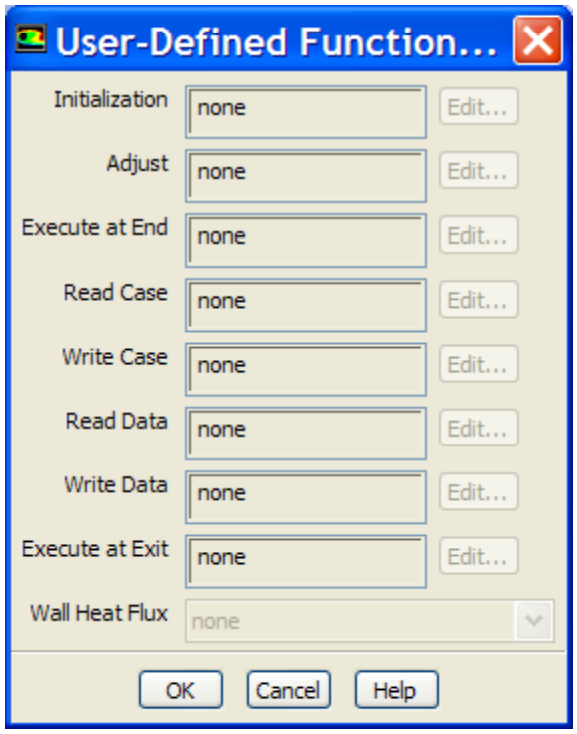
- The figure on the left shows the velocity field through the 2D elbow.
- The figure on the right shows the velocity vectors at the inlet. Notice the imposed parabolic velocity profile.





Other UDF Hooks

- 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
  - 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
- They are hooked into the solver using the User-Defined Function Hooks panel:



`Define > User Defined > Function Hooks`

Example 2 – Custom Initialization

- Initialize:
  - A temperature of 600 K
  - inside a sphere, with its center at (0.5, 0.5, 0.5),
  - radius of 0.25,
  - and 300 K throughout the rest of the domain.
- The domain pointer is passed to this UDF through the argument
- `thread_loop_c` macro is used to access all cell threads (zones), and `begin_c_loop` macro is used to access cells within each cell thread
- Deploy this UDF as a user defined function hook.

```
#include "udf.h"

DEFINE_INIT(my_init_function, domain)
{
 cell_t c;
 Thread *ct;
 real xc[ND_ND];
 thread_loop_c(ct,domain)
{
      begin_c_loop (c,ct)
    {
    C_CENTROID(xc,c,ct);
    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,ct) = 600.;
    else
       C_T(c,ct) = 300.;
    }
      end_c_loop (c,ct)
 }
}
```

DEFINE Macros

- Examples of top-level DEFINE macros

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

Geometry and Time Macros

`C_NNODES(c,t);`	Returns nodes/cell
`C_NFACES(c,t);`	Returns faces/cell
`F_NNODES(f,t);`	Returns nodes/face
`C_CENTROID(x,c,t);`	Returns coordinates of cell centroid in array `x[]`
`F_CENTROID(x,f,t);`	Returns coordinates of face centroid in array `x[]`
`F_AREA(A,f,t);`	Returns area vector in array `A[]`
`C_VOLUME(c,t);`	Returns cell volume
`C_VOLUME_2D(c,t);`	Returns cell volume (axisymmetric domain)

`CURRENT_TIME`	real current flow time (in seconds)
`CURRENT_TIMESTEP`	real current physical time step size (in seconds)
`PREVIOUS_TIME`	real previous flow time (in seconds)
`PREVIOUS_2_TIME`	real flow time two steps back in time (in seconds)
`PREVIOUS_TIMESTEP`	real previous physical time step size (in seconds)
`N_TIME`	integer number of time steps
`N_ITER`	integer number of iterations

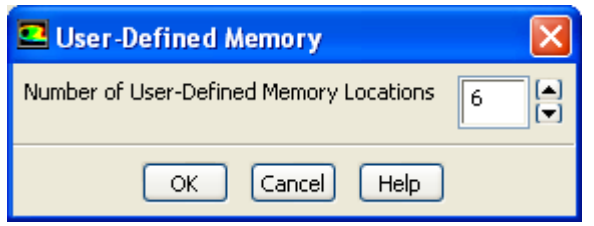
Cell Field Variable Macros

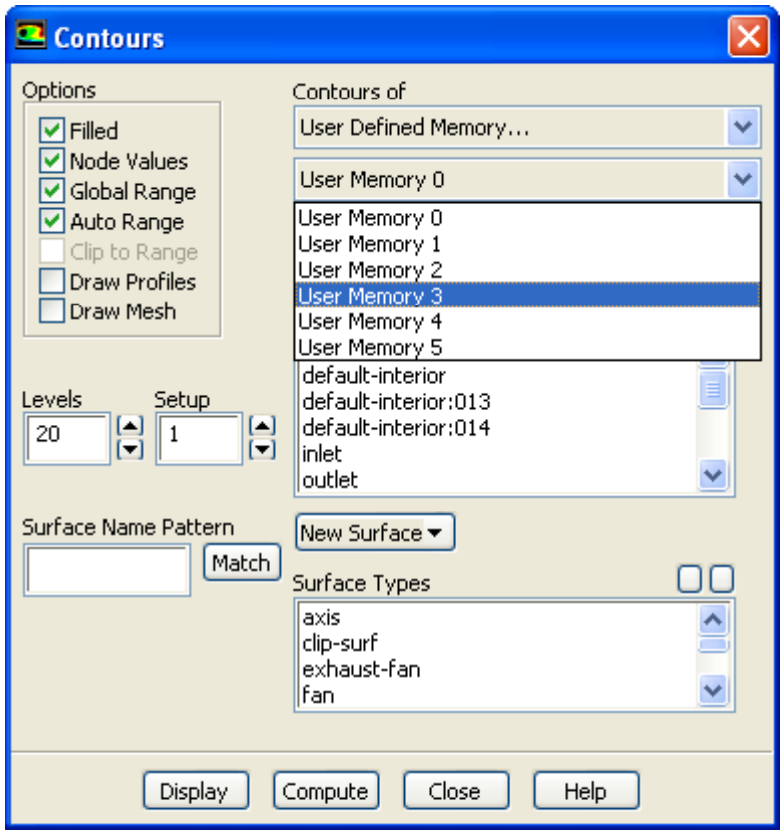
`C_R(c,t);`	Density
`C_P(c,t);`	Pressure
`C_U(c,t);`	U-velocity
`C_V(c,t);`	V-velocity
`C_W(c,t);`	W-velocity
`C_T(c,t);`	Temperature
`C_H(c,t);`	Enthalpy
`C_K(c,t);`	Turbulent kinetic energy (*k*)
`C_D(c,t);`	Turbulent dissipation rate (ε)
`C_O(c,t);`	Specific dissipation of *k* (ω)
`C_YI(c,t,i);`	Species 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

User Defined Memory

- User-allocated memory for each cell
  - 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 any purposes, including user's own numerical algorithms and postprocessing
  - Information is stored in the FLUENT data file.

Define → User-Defined → Memory



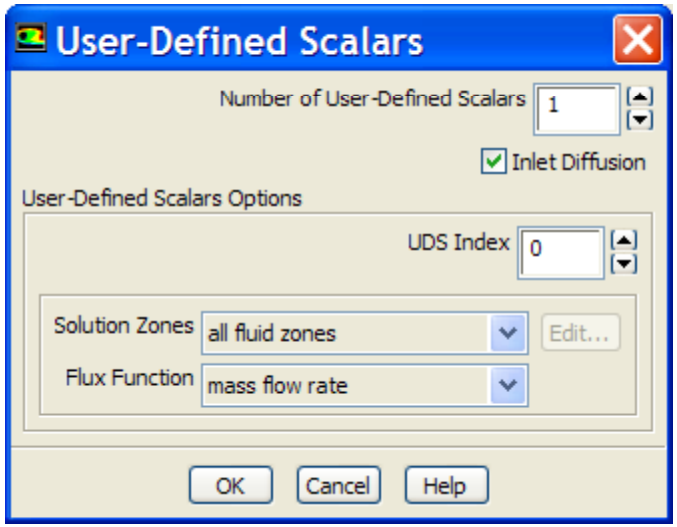


User Defined Scalars

- FLUENT can also solve transport equations for user-defined scalars

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

Define → User-Defined → Scalars



  - Number of UDS variables
  - In which zones the UDS is solved
  - Flux Function
    - `DEFINE_UDS_FLUX(name,face,thread,index)`
  - Unsteady function
    - `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 solve the electromagnetic field equations.

Additional Macros

- Still many more macros are available in the following categories and are documented in UDF Manual:
  - Turbulence models
  - Multiphase models
  - Reacting flows
  - Dynamic mesh
  - Input/Output

Where to Get More Information and Help

- UDF User Guide
  - Installed on your machine already as part of standard installation
  - Contains macro definitions, numerous code examples and code fragments.

- Start your own UDF program by modifying an existing UDF program which is close to what you want to do, then step by step add your own code to the program.

- Attend the Advanced UDF Training course

- Because UDFs can be very complicated, ANSYS does not assume responsibility for the accuracy or stability of solutions obtained using user-generated UDFs.

- Support will be generally be limited to guidance related to communication between UDFs and the FLUENT solver, we cannot help debug client's UDFs.