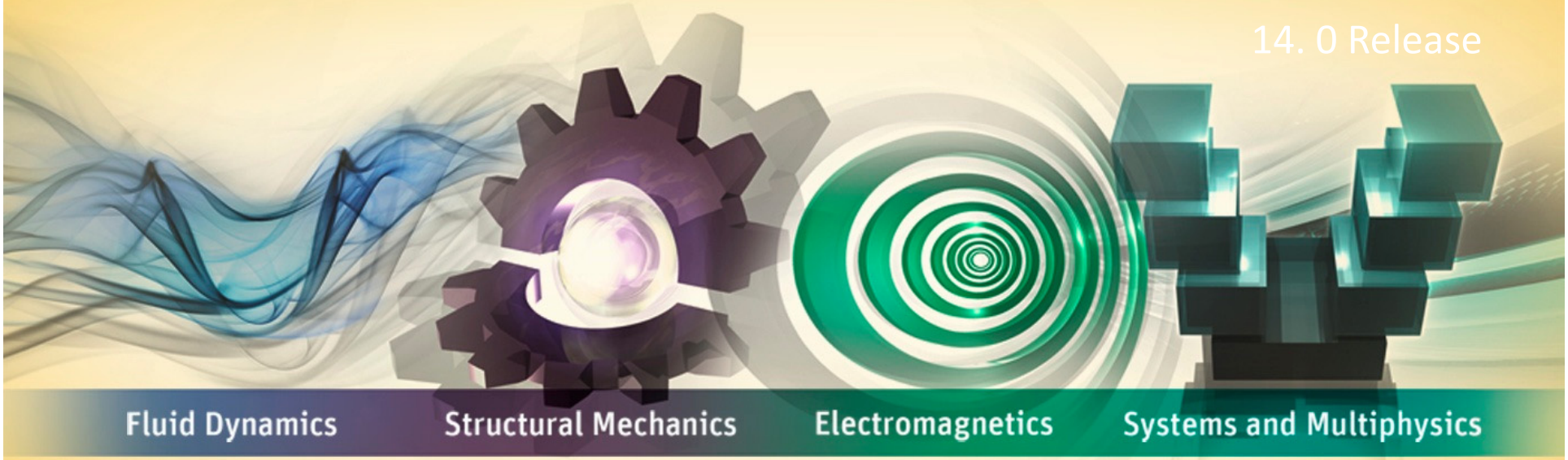
ANSYS

Realize Your Product Promise™

Lecture 11
User-Defined Functions

14. 0 Release



Introduction to ANSYS FLUENT

Introduction

Lecture Theme:

User-defined functions (UDFs) enable the customization of many aspects of a CFD model, including boundary and cell zone conditions, material properties, data output, and solution execution. Data exchange between the UDF and the solver takes place via pre-defined macros.

Learning Aims:

You will learn:

What UDFs can do

How to write and compile a basic UDF

How to hook UDFs to the FLUENT interface

Learning Objectives:

Almost all FLUENT users will end up having to write short UDFs from time to time. You will see how this can be done easily, even with only a rudimentary understanding of the C programming language.

UDF Overview

- 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
  - All data exchanged between the UDF and the solver must be in SI units

- 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

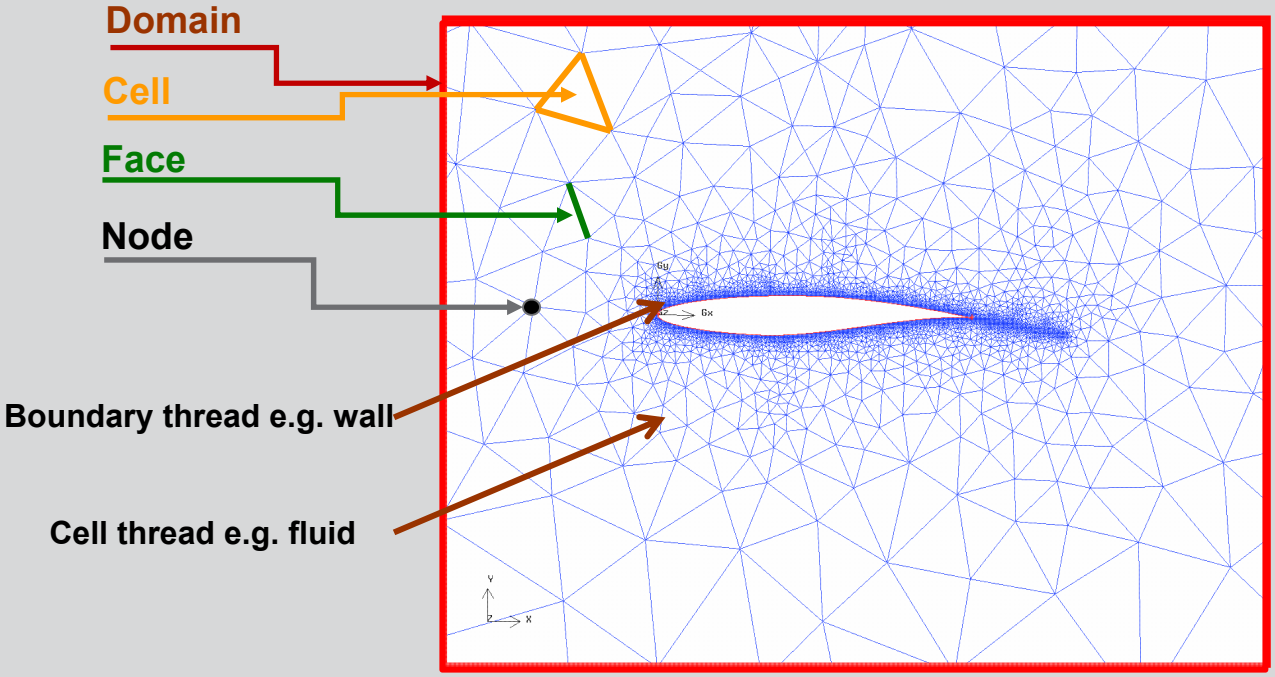
Interpreted vs. Compiled UDFs

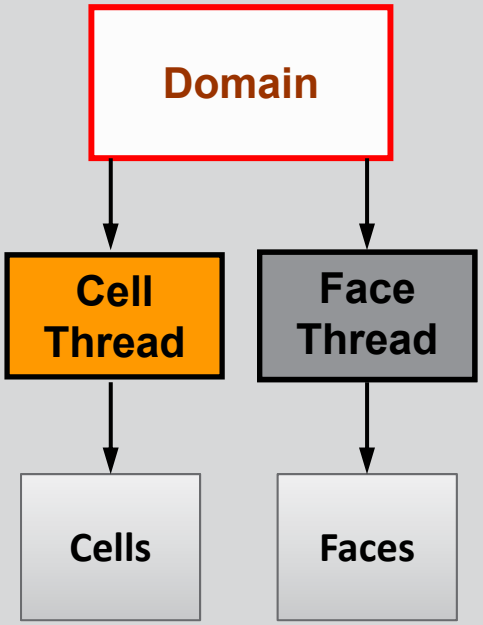
- UDFs can either be run compiled or interpreted.
  - The supported compiler for FLUENT on Windows platforms is Microsoft Visual Studio
  - Most Linux systems provide a C compiler as a standard feature.

- 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.

Fluent UDF Data Structure

- 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.

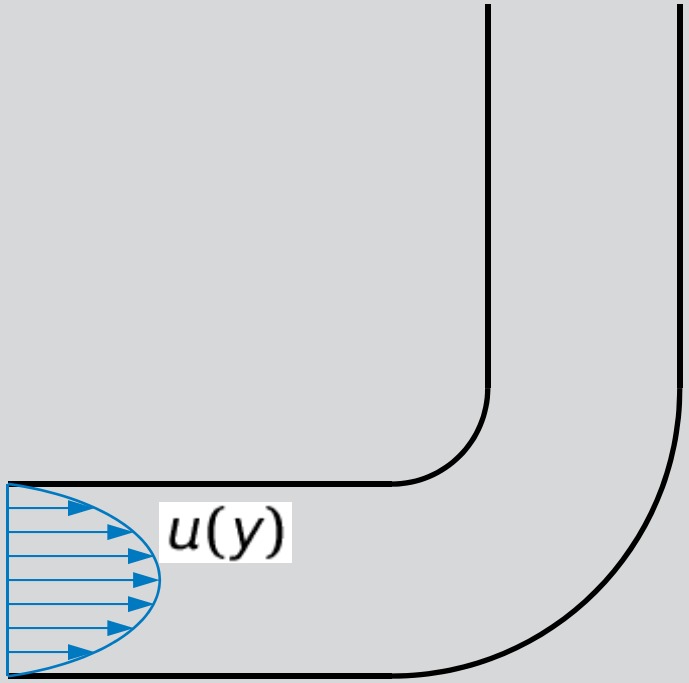
UDF Step by Step

- The basic steps for using UDFs in FLUENT are as follows :

1. Identify the problem to be solved
2. Check the usability & limitations of the standard models of FLUENT
3. Program the UDF (must be written in C)
4. Compile the UDF in the FLUENT session
5. Assign the UDF to the appropriate variable and zone in BC panel
6. Run the calculation
7. Examine the results

Example – Parabolic Inlet Velocity Profile

- **Example – Parabolic Inlet Velocity Profile**

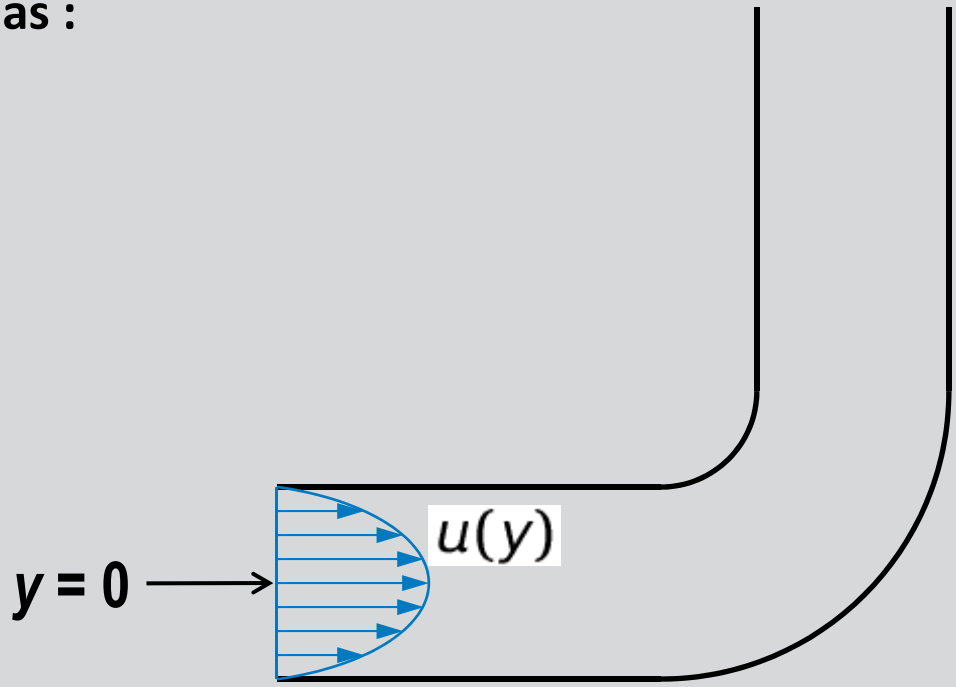


Step 1 – Identify the Problem to Solve

- **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 2 – Prepare the Source Code

- The **DEFINE_PROFILE** macro allows the function **x_velocity** to be defined.

- The code is stored as a text file **inlet_bc.c**
  - All UDFs begin with a **DEFINE_** macro
  - The name that will appear in the GUI will be **x_velocity**
  - The macro **begin_f_loop** loops over all faces **f**, pointed by thread
  - The **F_CENTROID** macro assigns cell position vector to **pos[]**
    - pos[0] is the x-coordinate
    - pos[1] is the y-coordinate
    - pos[2] is the z-coordinate
  - The **F_PROFILE** macro applies the velocity component to face **f,** using the desired arithmetic function

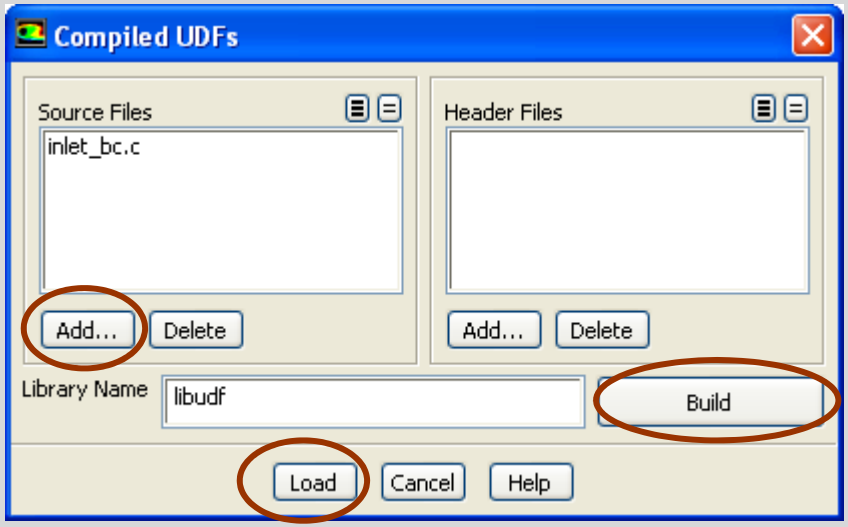
Header file "udf.h" must be included at the top of the program by the **#include** command

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

  begin_f_loop(f, thread)
  {
    F_CENTROID(pos,f,thread);
    y = pos[1];
    F_PROFILE(f, thread, nv)
          = 20.*(1.-
            y*y/(.0745*.0745));
  }
  end_f_loop(f, thread)
}
```

Step 3 – Compile the UDF in the FLUENT Session

- **Compiled UDF**

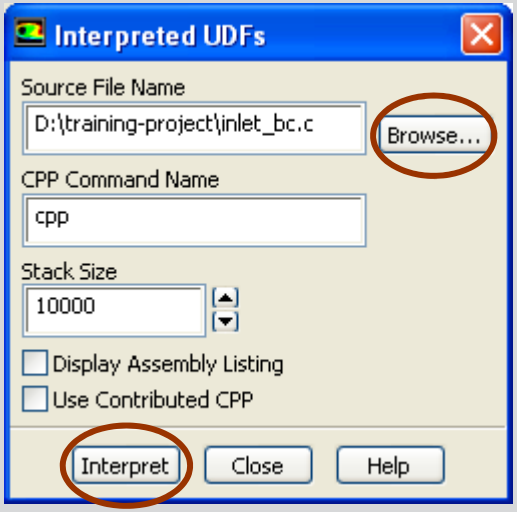
Define → User-Defined → Functions → Compiled



- 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**

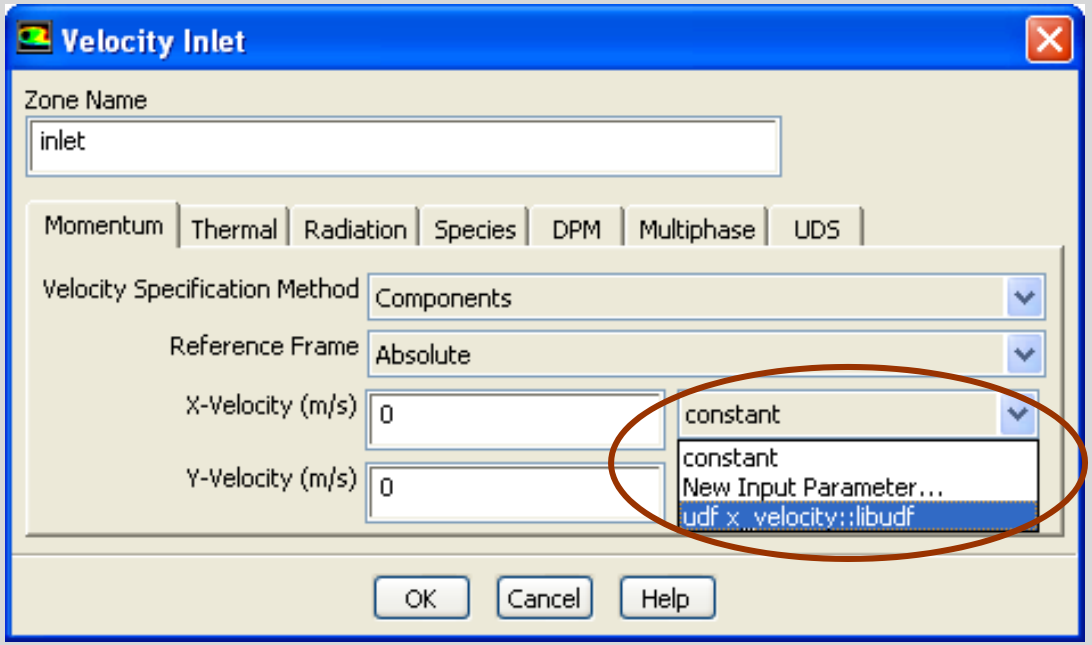
Define → User-Defined → Functions → Interpreted



- 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

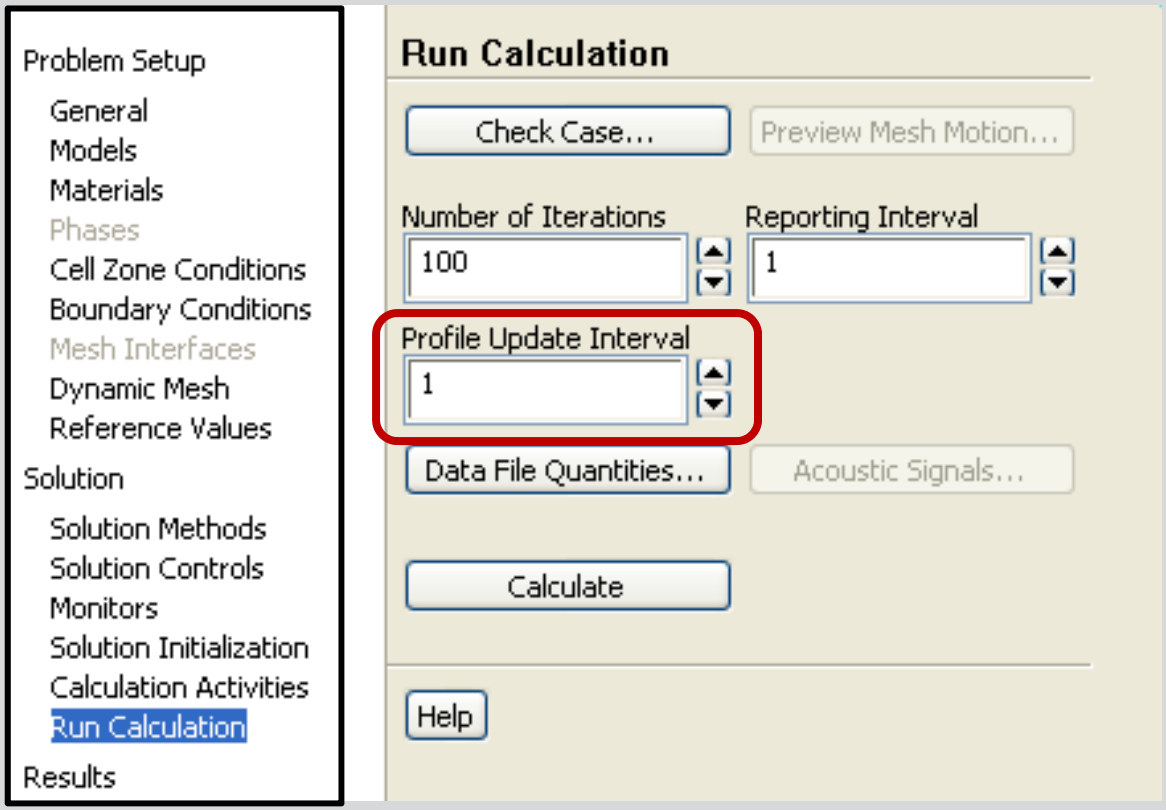
Step 4 – Hook the UDF in FLUENT GUI

- 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 5 – Run the Calculations

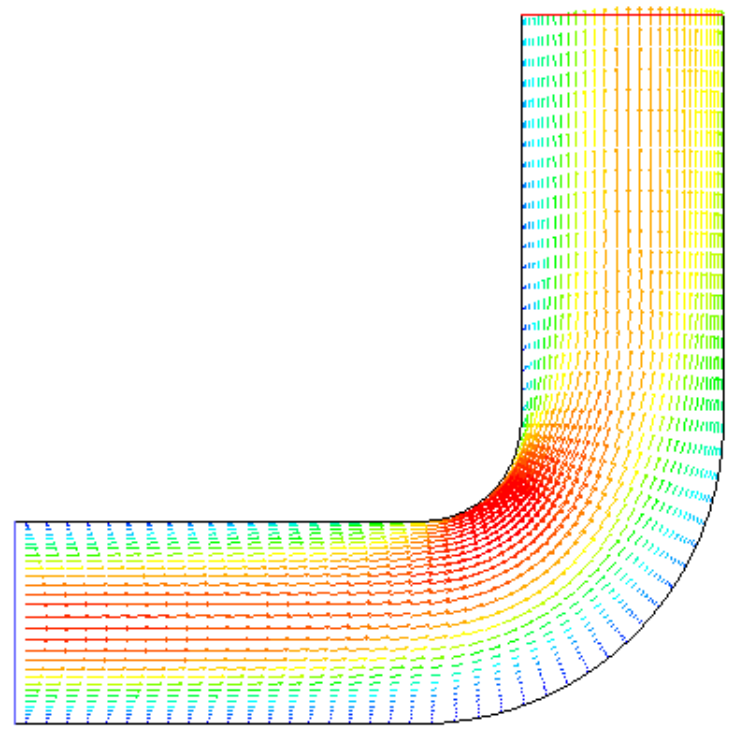
- You can change the Profile Update Interval in the Run Calculation panel (default value is 1)
  - This setting controls how often (either iterations or time steps if unsteady) the UDF profile is updated

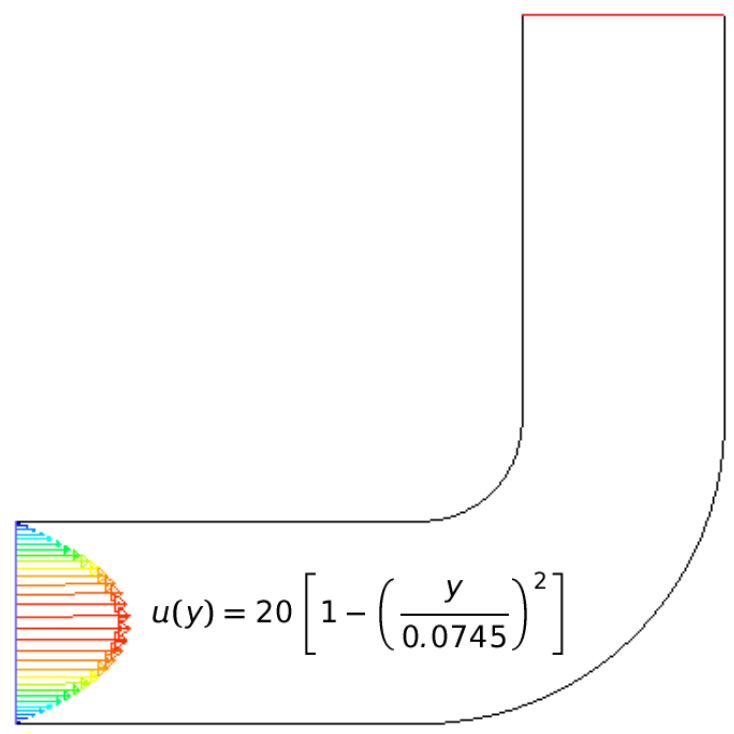


➢ … Run the calculation as usual …

Step 6 – Examine the Results

- 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





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
- C_T(c,ct) allows access to the cell temperature.
  - There are similarly named macros for all other solution variables
  - An extensive list of these macros can be found in the appendix
- Deploy this UDF as a user defined function hook in

  Define > User-Defined > Function Hooks (more details in Appendix)

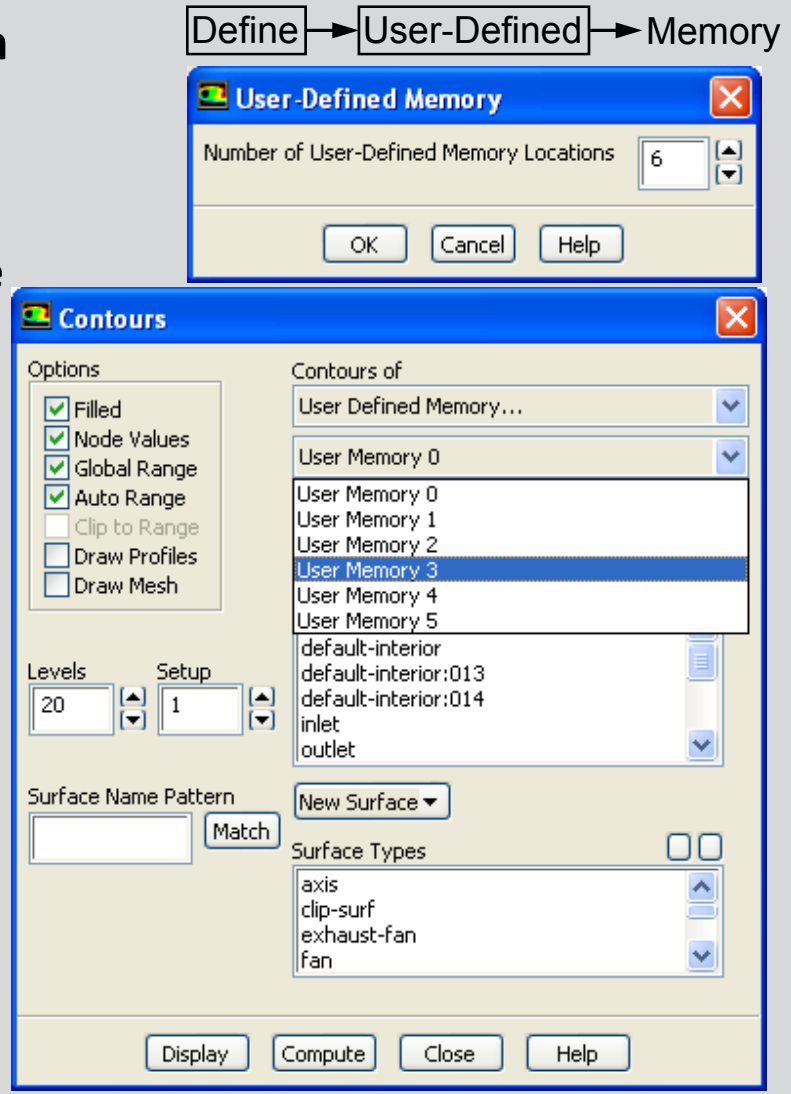
```
#include "udf.h"

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)
  }
}
```

Use of Macros in the Examples

- UDFs communicate with the solver through pre-defined macros
- Macros can be loosely categorized as either (with examples from the previous slides)
  - DEFINE macros
    - DEFINE_PROFILE(x_velocity,thread,nv), DEFINE_INIT (my_init_function,domain)
  - Looping macros
    - thread_loop_c (ct,domain){...}, begin_f_loop(f,thread){...}end_f_loop(f,thread)
  - Data access macros
    - Domain *d; thread *t; cell_t c; face_t f;
  - Geometry macros
    - C_CENTROID(xc,c,ct)
  - Solution data macros
    - C_T(c,t)
- More examples of each type of macro are described in the appendix
- Still many more DEFINE macros are available in the following categories and are documented in the UDF manual:
  - Turbulence models
  - Multiphase models
  - Reacting flows
  - Dynamic mesh
  - Input/Output

User Defined Memory

- **While macros are available for all solution variables, for instance C_T in Example 2, sometimes it can be helpful to create additional variables**
  - **For instance to store value of custom source terms, or custom post-processing variables**
- **This can be done with User-Defined Memory variables, which are user-allocated memory for each cell**
  - Up to 500 field variables can be defined
  - The UDF macros for cell values and face values, respectively, are
    - C_UDMI(cell,thread,index);
    - F_UDMI(face,thread,index);
  - Information is stored in the FLUENT data file and can be accessed in post-processing operations such as contour plots

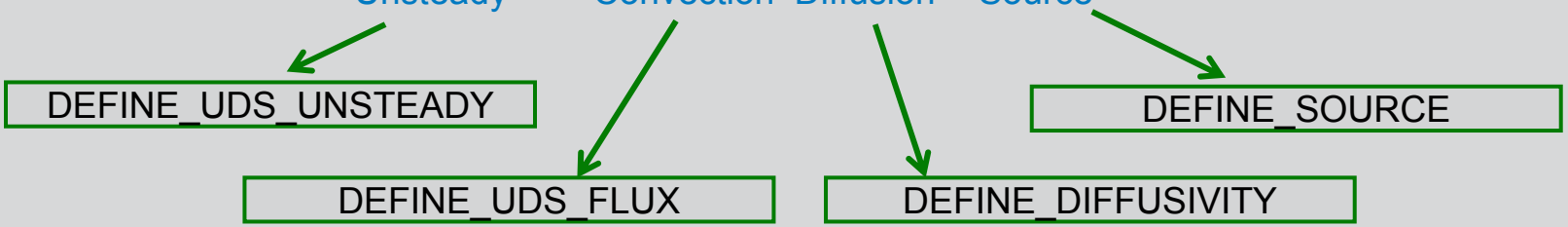


User Defined Scalars (UDS)

- FLUENT can solve up to 50 generic transport equations for user-defined scalars
  - The UDS equations can be solved as a standard steady or transient convection-diffusion transport equation, or they can be customized through UDFs, to represent other partial differential equations, for instance the electromagnetic field equations
    - All terms in the equations can be controlled through UDFs

$$\underbrace{\frac{\partial \phi_k}{\partial t}}_{\text{Unsteady}} + \underbrace{\frac{\partial}{\partial x_i}\left(F_i \phi_k}_{\text{Convection}} \underbrace{- \Gamma_k \frac{\partial \phi_k}{\partial x_i}\right)}_{\text{Diffusion}} = \underbrace{S_{\phi_k}}_{\text{Source}}$$

$$k = 1, 2, \ldots, N_{\text{scalar}}$$



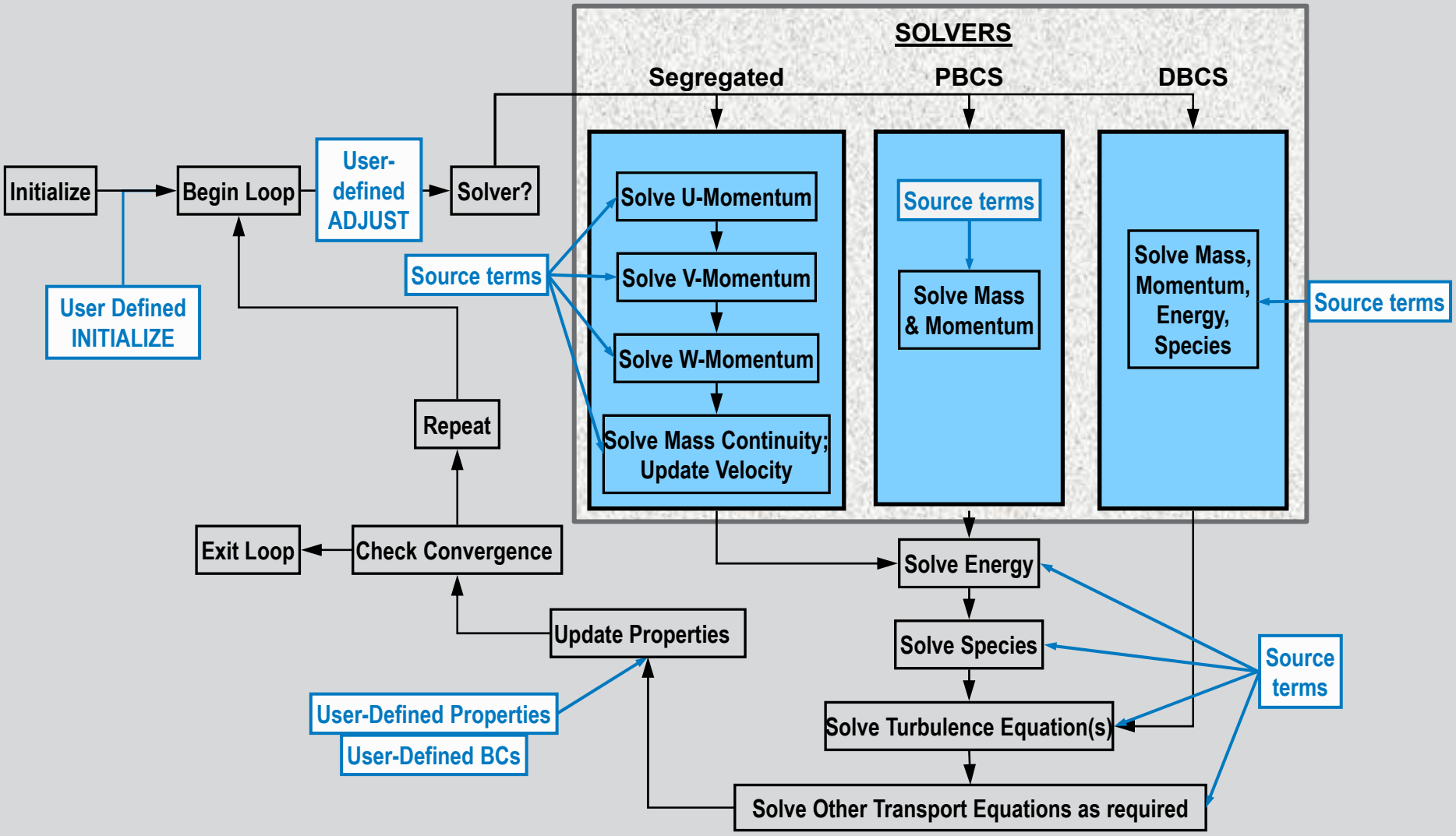
Where to Get More Information and Help

- 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.

ANSYS

Appendix

User Access to the FLUENT Solver



FLUENT Data Structure

- In order to access data in a thread (zone), we need to :
  - Provide the correct Thread pointer :

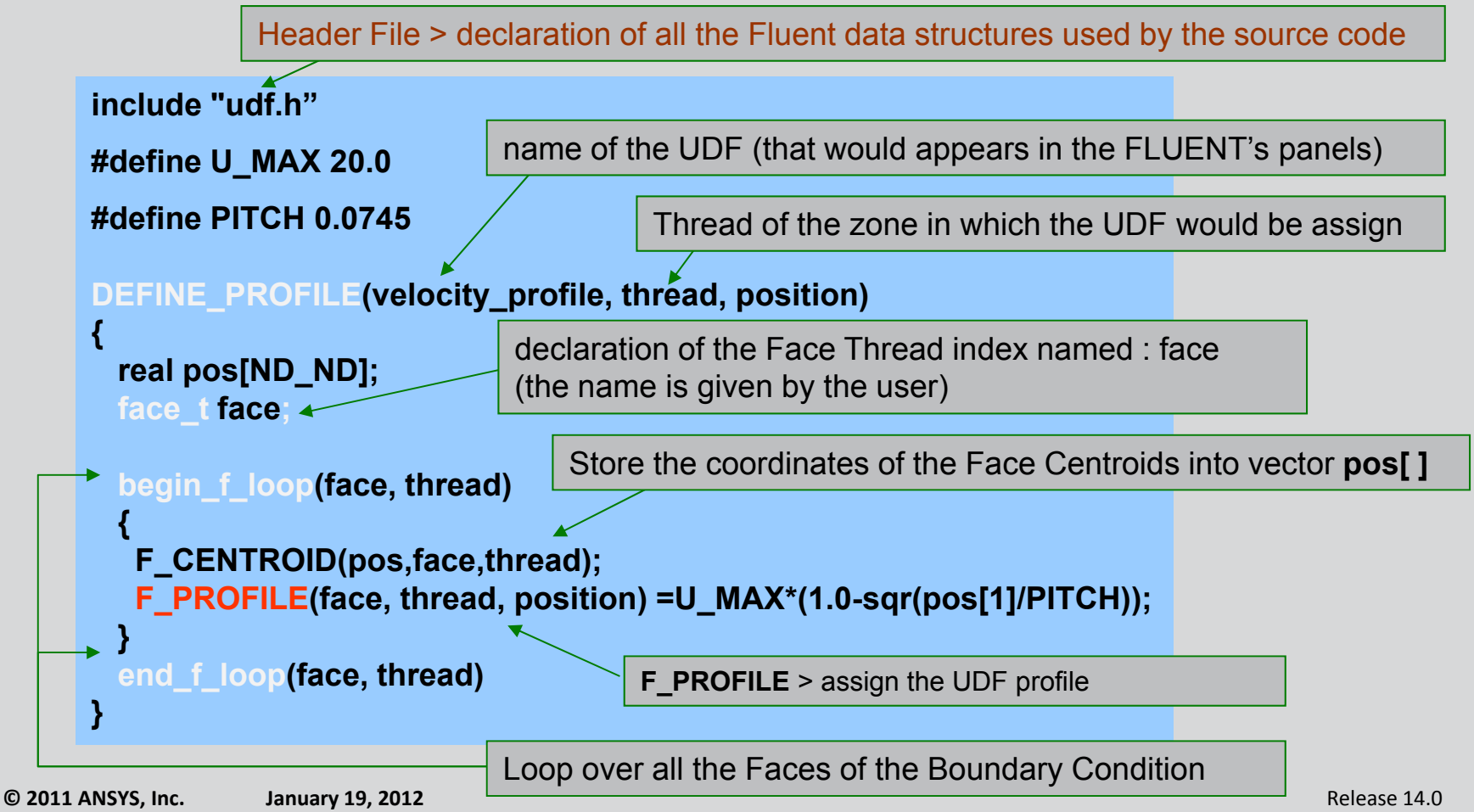
Type	Variable	Meaning of the declaration
`Domain`	`*d;`	`d` is the user name of a pointer to domain Thread
`Thread`	`*t;`	`t` is the user name of a pointer to Thread
`cell_t`	`c;`	`c` is the user name of a Cell thread variable
`face_t`	`f;`	`f` is the user name of Face thread variable
`Node`	`*node;`	`node` is the user name of a pointer to a Node

  - use FLUENT provided loop macro to access each member (cell or face) in that Thread :
    - `thread_loop_c(ct,d) { }` **Loop over all cell threads in domain** `d;`
    - `thread_loop_f(ft,d) { }` **Loop over all face threads in domain** `d;`
    - **Loop over all cells in a cell thread** `t;`

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

Step 3 : Program the UDF

- To program the UDF, we need to use two Macros :
  - one to perform the boundary condition assignment (DEFINE_PROFILE),
  - and another to loop over the centroids of the inlet faces (begin_f_loop),



Overview of FLUENT Macros

- 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

DEFINE_ Macros

- Any UDF you write MUST begin with a `DEFINE_` macro

- Most often used 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_PROFILE(name,thread,index);` boundary profiles

`DEFINE_SOURCE(name,cell,thread,dS,index);` equation source terms

`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

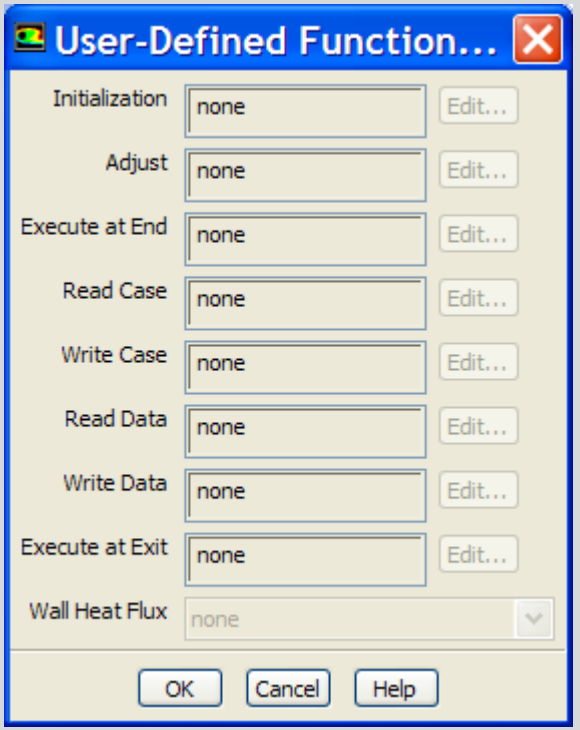
`DEFINE_UDS_UNSTEADY(name,cell,thread,index,apu,su);` UDS transient terms

`DEFINE_DELTAT(name,domain);` variable time step size for unsteady problems

DEFINE_ Macros - UDF Hooks

- **Some DEFINE_ macros are hooked into the solver using the User-Defined Function Hooks panel**
  - **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**

Define → User-Defined → Function Hooks



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

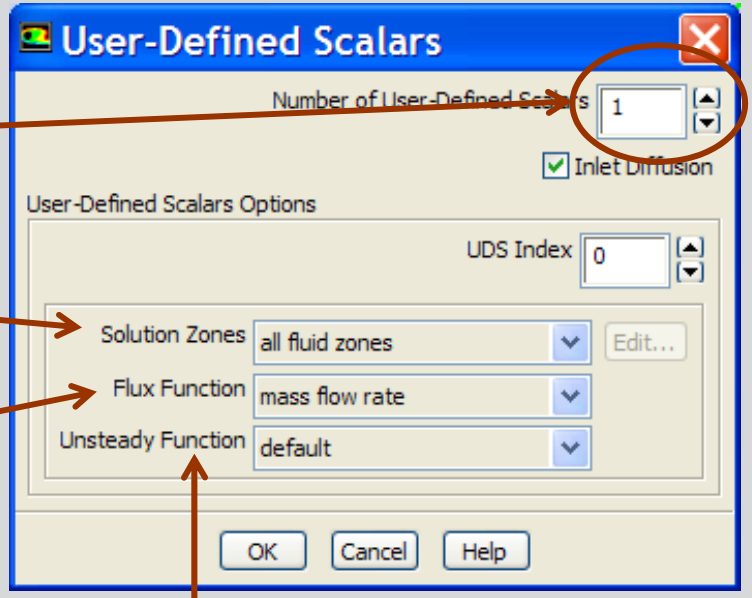
Cell Field Variable Macros

`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 Scalars

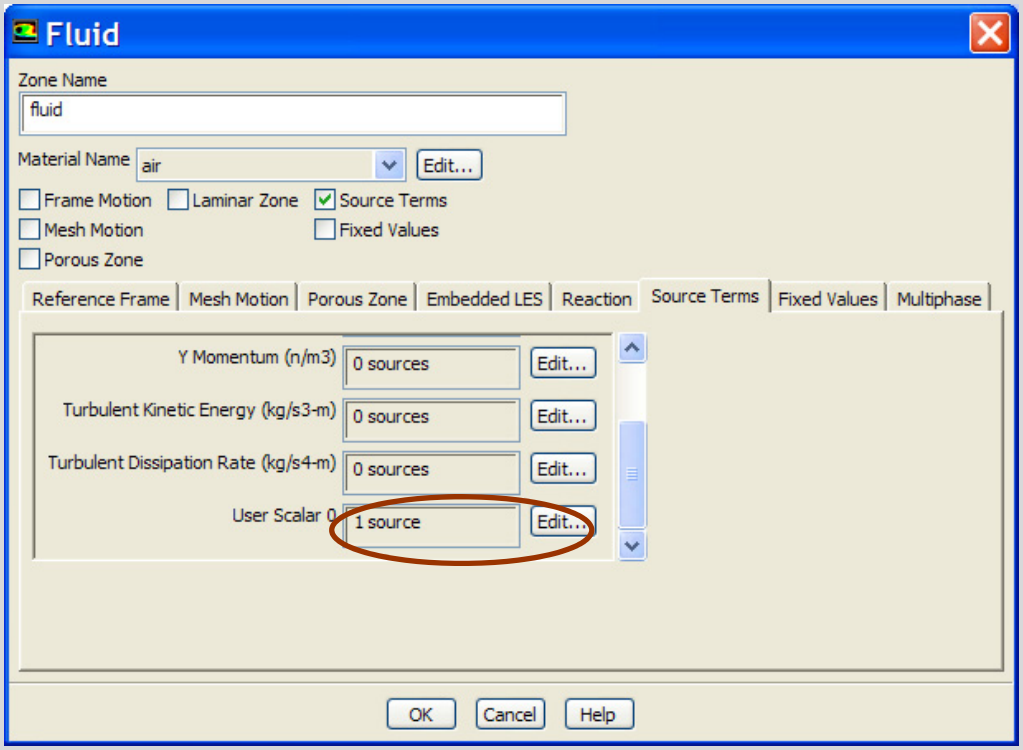
- **UDS Macros Hooks (1) :**
  - **Number of UDS variables**
  - **Zones in which the UDS is solved**
  - **Flux Function**
    - `DEFINE_UDS_FLUX(name,face,thread,index)`
  - **Unsteady function**
    - `DEFINE_UDS_UNSTEADY(name,cell,thread,index,apu,su)`
    - **If statements are required in order to associate multiple flux and transient functions with each UDS**

Define → User-Defined → Scalars



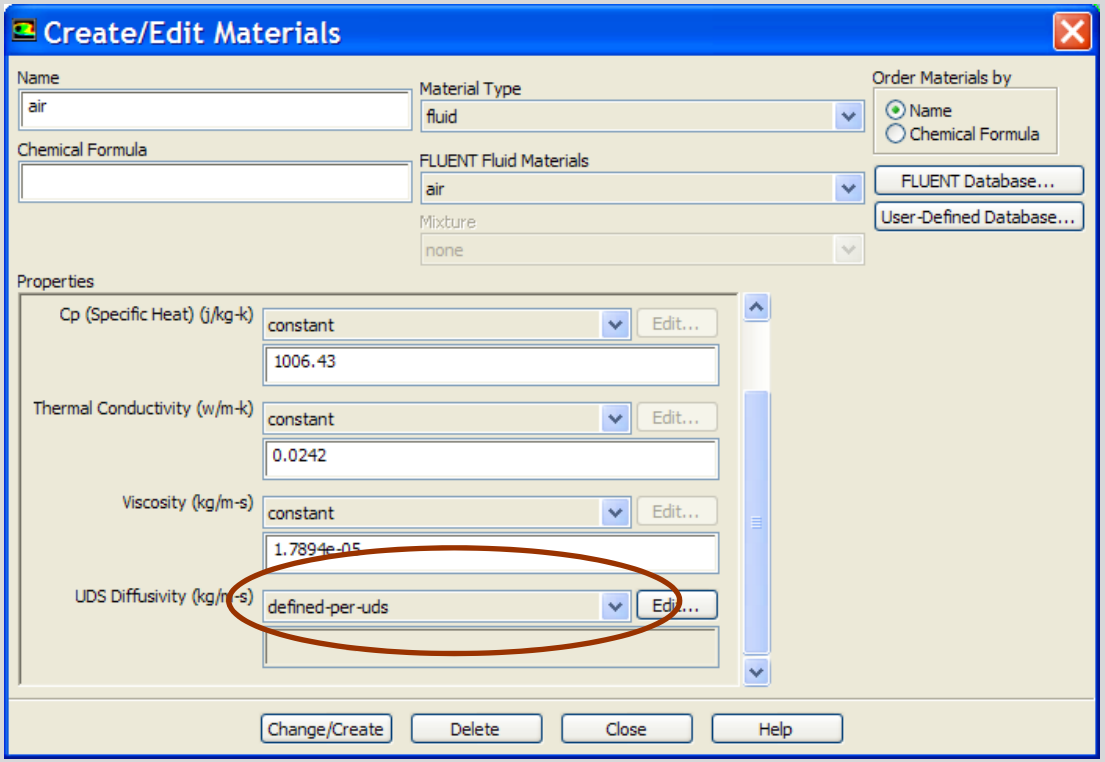
User Defined Scalars

- **UDS Macros Hooks (2) :**
  - **Source term**
  - `DEFINE_SOURCE(name,cell,thread,dS,index)`



User Defined Scalars

- **UDS Macros Hooks (3) :**
  - **Diffusivity**
    - `DEFINE_DIFFUSIVITY(name,cell,thread,index)`



User Defined Scalars

- **UDS Macros Hooks (4) :**
  - **Boundary Conditions**
    - **Specified Flux or Specified Value**
    - **Define as constant or with UDF**

– `DEFINE_PROFILE(name,thread,index)`