Tutorial: UDFs For A User-Defined Scalar

1 Introduction

FLUENT solves the transport equation for a user-defined scalar (UDS) in the same way as it solves the transport equation for a scalar in the core equations, such as a species mass fraction. The UDS capability can be used to implement a wide-range of physical models in magnetohydrodynamics, electromagnetics, and more.

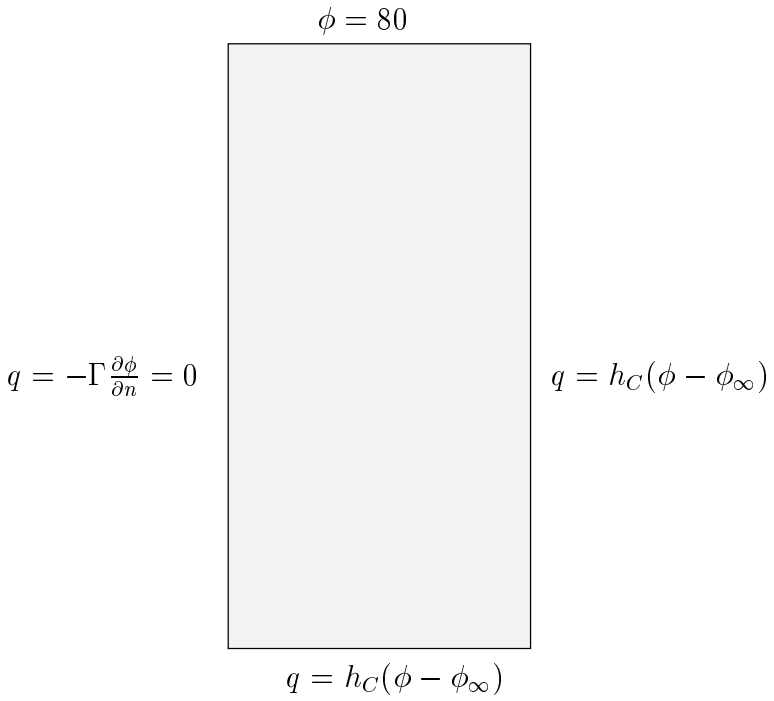
In this tutorial we will learn to solve a general scalar diffusion equation as follows:

$$c\frac{\partial \phi}{\partial t} - \nabla \cdot (\Gamma \nabla \phi) = S_\phi, \qquad \text{in } \Omega,\ t > 0 \tag{1}$$

with the following possible types of boundary condition (BC) at the boundary (or a part of the boundary) of the domain:

- Dirichlet BC: $\phi = D_0$
- Neumann BC: $-\Gamma(\partial\phi/\partial n) = q_0$
- Mixed BC: $-\Gamma(\partial\phi/\partial n) = h_C(\phi - \phi_\infty)$

Here, D_0, q_0, h_C and ϕ_∞ are constant values.



2 The Steady-state Solver

First, we tackle a steady-state scalar equation with constant Γ and zero source term (S_ϕ=0) as follows:

$$-\nabla \cdot (\Gamma \nabla \phi) = 0 \tag{2}$$

This is the Laplace's equation.

Solving the Laplace's equation by the FLUENT UDS solver does not necessarily require user-defined functions (UDFs), one can simply activate the UDS from the graphical user interface. But FLUENT UDS only provides Dirichlet and Neumann conditions for the boundaries, hence we need to use a UDF in order to apply the *mixed* boundary condition for a user-defined scalar equation.

2.1 Formulation of the Mixed Boundary Condition

For a generic cell $c0$ adjacent to the boundary, the diffusive flux across the boundary face f of the cell is expressed as follows:

$$-\int_f \Gamma \left(\frac{\partial \phi}{\partial n}\right) dS = \int_f h_C(\phi - \phi_\infty)\, dS \tag{3}$$

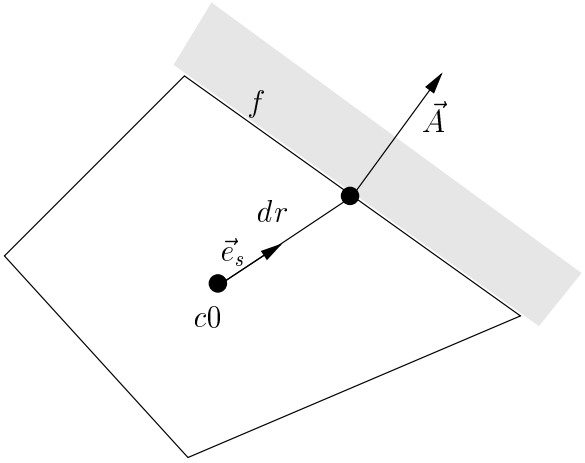
Use the mid-point rule of surface integral, the diffusive flux can be approximated by

$$-\Gamma_f \left(\frac{\partial \phi}{\partial n}\right)_f A_f = h_C(\phi_f - \phi_\infty) A_f \tag{4}$$

In the FLUENT solver, the diffusive flux $\Gamma_f(\partial\phi/\partial n)_f A_f$ is approximated by two parts: the "primary" gradient is evaluated implicitly along the line connecting the cell centroid $c0$ to the centroid of face f, then it is corrected by a *secondary* gradient (or cross diffusion) term evaluated explicitly by the gradient obtained from the previous iteration ($\overline{\nabla\phi}$):

$$\Gamma_f \left(\frac{\partial \phi}{\partial n}\right)_f A_f \approx \underbrace{\Gamma_f \frac{(\phi_f - \phi_{c0})}{dr}\left(\frac{\vec{A}\cdot\vec{A}}{\vec{A}\cdot\vec{e_s}}\right)}_{\text{primary}} + \underbrace{\Gamma_f \left(\overline{\nabla\phi}\cdot\vec{A} - \overline{\nabla\phi}\cdot\vec{e_s}\frac{\vec{A}\cdot\vec{A}}{\vec{A}\cdot\vec{e_s}}\right)}_{\text{secondary}} \tag{5}$$

FLUENT provides users with a macro which defines the necessary geometrical variables of the cell, and another macro which calculates the secondary gradient term in Eq.(5):




```
BOUNDARY_FACE_GEOMETRY(f, t, A, ds, es, A_by_es, dr0)
BOUNDARY_SECONDARY_GRADIENT_SOURCE(source,SV_UDSI_G(i),dG,es,A_by_es,k)
```

If we designate the secondary gradient term in Eq. (5) as β_0, and $(\vec{A}\cdot\vec{A})/(\vec{A}\cdot\vec{e_s})$ as A_{be}, Eq. (5) and Eq. (4) can be written as:

$$-h_C(\phi_f-\phi_\infty)A_f=\Gamma_f\frac{(\phi_f-\phi_{c0})}{dr}A_{be}+\beta_0 \tag{6}$$

and ϕ_f can be expressed as follows:

$$\phi_f=\frac{\Gamma_f(A_{be}/dr)\phi_{c0}-\beta_0+h_CA_f\phi_\infty}{h_CA_f+\Gamma_f(A_{be}/dr)} \tag{7}$$

The mixed boundary condition for a user-defined scalar is ready to be specified by boundary profile ϕ_f in Eq. (7) through the UDF macro `DEFINE_PROFILE()`. The source code is listed in Appendix A of this tutorial.

2.2 FLUENT Case Setup for the Steady-state UDS Solver

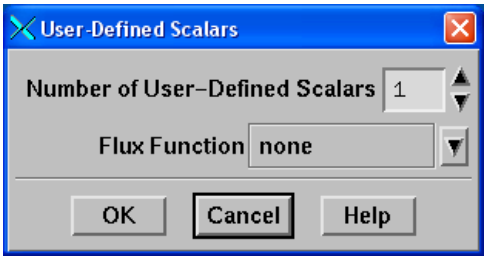
As shown in the problem illustration, the problem is a 2-D rectangle, with constant flux, constant value and mixed boundary conditions along the boundary edges.

For the problem, Γ=0.162 W/(m K), specific heat of the material is 1650 J/(kg K), and ϕ_∞=550°C, h_C=20 W/(m^{2}K).

1. Start `fluent 2d`. Read in the mesh file `laplace.msh` and perform the grid check.

2. Keep the default solver settings: 2-D, steady-state, segregated solver.

3. Turn on the user-defined scalar:
   Define $\longrightarrow$ User-Defined $\longrightarrow$ Scalars.
   Increase the number of UDS to 1, and leave the Flux Function as none. Click on OK to close the panel.



4. Keep the default Operating Conditions.

5. Compile the UDF source code: In the compiled UDF panel, click Add on the left, select `mixedbc.c`, then click Build. The compilation is done when it is indicated in the FLUENT main console. Finally click Load to load the library.

6. Set up the following material properties for the problem: UDS diffusivity (Γ) is set as a constant at 0.162. Since we will not use density nor viscosity in this example (see Eq. (2)), we can set them to any value. Then change the material name to maple, and save it over air. Note that in this example, consistency of units is maintained strictly by us as users according to the physics of the problem.

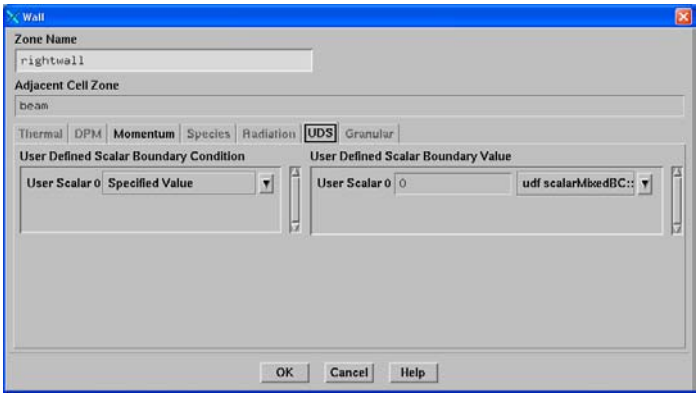
7. Select the boundary conditions as follows:

   - Top wall: constant value at 80.

- Left wall: zero flux (specified flux at constant value 0).
- Right and bottom walls: mixed boundary condition:

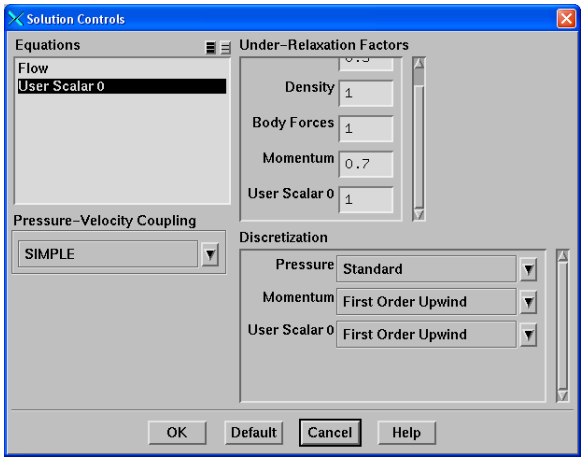
$$-\left(\Gamma\frac{\partial\phi}{\partial n}\right)_w = h_C(\phi - \phi_\infty)$$

where Γ is the UDS diffusivity (0.162), h_C=20 and ϕ_∞ is 550. The panel shot is shown for the `rightwall`: specified value BC is supplied by the appropriate UDF hook (`udf scalarMixedBC`).



- Make sure `maple` is the material assigned to the whole fluid domain (`beam`) (Note: in FLUENT 6.2, UDS solves only the fluid region).

8. Turn off the Flow Equations in Solve $\longrightarrow$ Controls $\longrightarrow$ Solution. Keep the default settings for the rest.



9. Turn on monitor residual plot, and turn off "Check for Convergence" for uds-0.

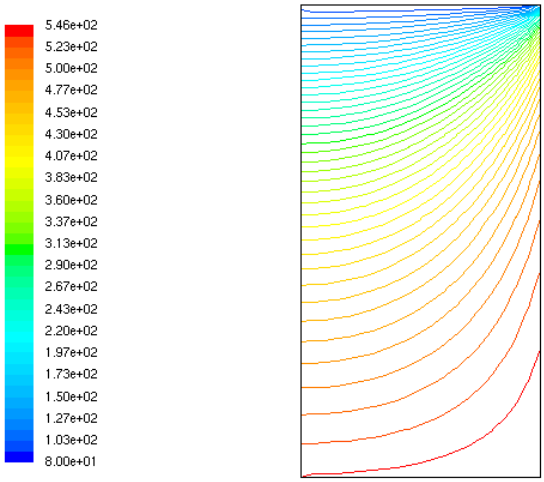
10. In Surface $\longrightarrow$ Point, make a point in the center (x=0.025, y=0.05) of the rectangle for monitoring and call it the `middlepoint`.

11. In Monitors $\longrightarrow$ Surface, turn on the monitoring (plot and print) of the Sum of user-scalar-0 on `middlepoint` per iteration. Monitoring the variation of ϕ at this point can help us decide whether the calculation is converged or not.

12. Start the solution for about 50 iterations. The value of ϕ no longer changes after about 40 iterations, and we consider the solution is converged. Save the case and data file as `laplace.cas/dat`.

2.3 Results

Here we use Display $\longrightarrow$ contours to visualize the solution of the problem. As it is clear in the contour plot of ϕ, the zero flux boundary on the left side is equivalent to a symmetry boundary condition.



3 The Unsteady Solver

The unsteady user-defined scalar equation

$$c\frac{\partial \phi}{\partial t} - \nabla \cdot (\Gamma \nabla \phi) = 0, \qquad \text{in } \Omega,\ t > 0 \tag{8}$$

is to be solved with the same boundary conditions and a given initial condition.

When the transient term is $\partial(\rho\phi)/\partial t$, the FLUENT solver can readily handle this unsteady UDS term when the unsteady solver is turned on (either first- or second-order). But if the transient term of the user's equation is not the same as the given form, users must supply the unsteady term through the `DEFINE_UDS_UNSTEADY( )` macro. For example, in Eq. (8), c is a variable. Note that for an unsteady heat conduction problem in dimensional form, c in Eq. (8) represents ρc_p of the solid material.

3.1 First-order Unsteady Formulation

In finite-volume methods, the transient term is first approximated by a first- or second-order finite-difference expression, then integrated with respect to the cell volume. The FLUENT solver expects this transient term to be moved to the right-hand side of the governing equation and included in the discretized equation as a source term.

The first-order finite-difference backward differencing approximation gives

$$\frac{\partial \phi}{\partial t} \approx \frac{\phi^n - \phi^{n-1}}{\Delta t}$$

where ϕ^n indicates the value of ϕ at the current time level, ϕ^{n-1} is the value at the previous time level, and Δt is the time-step size. Hence

$$-\int c\frac{\partial \phi}{\partial t}dV \approx \underbrace{\left(-c\frac{\Delta V}{\Delta t}\right)}_{A_{pu}}\phi^n + \underbrace{c\frac{\Delta V}{\Delta t}\phi^{n-1}}_{S_u} \tag{9}$$

where ΔV represents the volume of each individual cell. In Eq. (9), the first term on the right-hand side (RHS) represents the *implicit* part, and the coefficient multiplying ϕ^n is denoted as A_{pu}; the second term on the RHS, called S_u, is the explicit part because it is expressed by known values at the previous time-step. We can rewrite the volume integral of the unsteady term as:

$$-\int c\frac{\partial \phi}{\partial t}dV = A_{pu}\,\phi^n + S_u \tag{10}$$

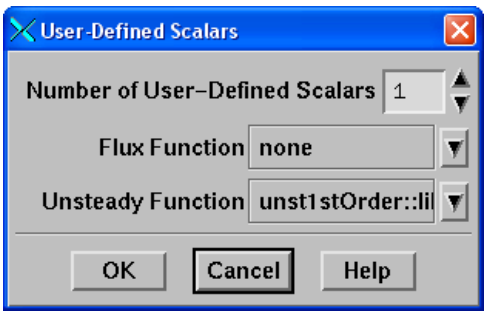
and it is ready to be coded in the unsteady macro `DEFINE_UDS_UNSTEADY()`. The code is listed in Appendix B.

3.2 FLUENT Case Setup for the Unsteady UDS Solver

The Unsteady diffusion problem represented by Eq. (8) over the same rectangular domain is solved by a first-order implicit formulation. The boundary condition is also unchanged. The initial condition is $\phi = 80$ throughout the domain.

Since the problem is partially set up in the steady case, here we only outline the unique steps for the setup of the unsteady solver:

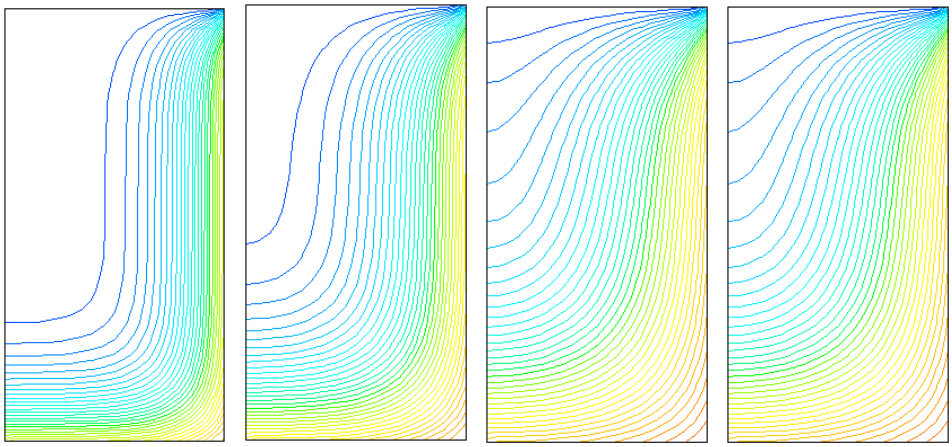
1. Read in the case file of the steady-state problem `laplace.cas`.
2. Turn on the unsteady term by selecting first-order implicit formulation in Define $\longrightarrow$ Solver.
3. Compile and load the source code `transientMixedBC.c` in Define $\longrightarrow$ User-defined $\longrightarrow$ Functions $\longrightarrow$ Compiled.
4. Select the first-order unsteady UDF by opening Define $\longrightarrow$ User-defined $\longrightarrow$ Scalars and select `unst1stOrder::libudf` in the panel:



5. Initialize the solution by setting scalar-0 to be 80.
6. In the Iterate panel, set time-step size to 0.1 s, and set the number of iterations per time step to 40. Iterate for 200 time steps.

3.3 Results

Contours of ϕ when t=1, 2, 3, 4 s, respectively, are plotted in the following figure. The diffusion of ϕ into the domain due to high ϕ_∞ in the ambient is clearly visible via the mixed (convective) boundary condition.



$t = 1$ s. $t = 2$ s. $t = 3$ s. $t = 4$ s.

Besides the contour plot, profiles and fluxes of ϕ versus time can be used to investigate the numerical solution of the model problem.

4 Second-order Unsteady Formulation

It can be easily shown that the following finite-difference, backward differencing approximation to $\partial\phi/\partial t$ is second-order accurate in time:

$$\frac{\partial \phi}{\partial t} \approx \frac{3\phi^n - 4\phi^{n-1} + \phi^{n-2}}{2\Delta t} \tag{11}$$

It involves the values of ϕ at three different time levels: ϕ^n, ϕ^{n-1} and ϕ^{n-2}.

You can use the first-order implicit UDF code in the appendix as a useful guide to come up with your own second-order implicit unsteady UDF code.

A few hints for the exercise:

- In the *first step* of the unsteady simulation ($n = 1$), ϕ^{n-2} is not available yet (ϕ^{n-1} is the initial condition). Therefore we can use the first-order formulation in this step in order to advance to the second time step ($n = 2$).
- ϕ^{n-2} is represented by `C_UDSI_M2(c,t,i)` for each cell.
- Get the signs right — remember that the transient term is moved to the RHS of the equation as a source term.
- Before starting the iteration, the second-order unsteady formulation must be turned on in Define $\longrightarrow$ Solver.

5 Non-constant Source Term

Another exercise is to implement a non-constant source term in Eq. (1). The macro `DEFINE_SOURCE(name, c, t, dS, eqn)` is called to represent S_ϕ. The finite-volume

solver of FLUENT expects the source term to be *linearized* according to the following convention:

$$S_\phi = A + B\phi = \underbrace{\left(S^* - \left(\frac{\partial S_\phi}{\partial \phi}\right)^* \phi^*\right)}_{A} + \underbrace{\left(\frac{\partial S_\phi}{\partial \phi}\right)^*}_{B} \phi \tag{12}$$

where the superscript $*$ represents the value at the previous iteration, hence A (called `source`) and B (called `dS[eqn]`) can be coded explicitly by using currently known value of ϕ. Note that `DEFINE_SOURCE( )` is a general UDF for all variables. To use it for a UDS (ϕ), one needs to hook it up in the boundary condition panel to the cell zone where ϕ is to be solved.

There are various ways to linearize a source term, but the general requirement is that B (the slope) should be *non-positive* to enhance convergence of the iterative solution process. Actually, the more negative the slope is, the better.

6 Summary

A user-defined scalar diffusion equation has been modeled by using UDFs in FLUENT. Some details of the finite-volume method used by the solver were carefully discussed when implementing terms in the governing equation and BC. However, treatment of the advective term $\nabla \cdot (\vec{F}\phi)$ ($\vec{F}$ is the general flux vector), is not covered here. It will be tackled in a related tutorial.

Appendix A: Mixed Boundary Condition for a UDS

```
/************************************************************************/
/* Implementation of the mixed boundary condition for a UDS (or multiple): */
/*       q = hC ( phi - PHI_inf )                                       */
/************************************************************************/

#include "udf.h"
#include "sg.h"  /* needed for the boundary and secondary gradient macros */

/*======================================================================*/
#define CP 1650.0  /* heat capacity for Maple in (J/kg K) */
#define HTC 20.0   /* heat transfer coefficient for the problem */
#define TINF 550   /* ambient temperature of the problem */
/*======================================================================*/


/* Names of the user-defined scalar to be used */
enum
  {
    phi1,
    N_REQUIRED_UDS
  };


DEFINE_PROFILE(scalarMixedBC, thread, nv)
{
  /* constants must be specified correctly for the mixed BC */
  real hC, PHI1_inf;

  /* ================== */

  face_t f;

  real A[ND_ND], dG[ND_ND], dr0[ND_ND], es[ND_ND], dr, A_by_es;
  real Af;
  real beta0, gamma;
  real temp1, temp2;
  Thread *t0=thread->t0;

  hC=HTC;
  PHI1_inf=TINF;

  begin_f_loop(f, thread)
    {

     /* identify the cell thread adjacent to the face thread f */
      cell_t c0 = F_C0(f, thread);

      BOUNDARY_FACE_GEOMETRY(f, thread, A, dr, es, A_by_es, dr0);
      Af=NV_MAG(A);
      gamma=C_UDSI_DIFF(c0, t0, phi1);
```

```
      if (NULLP(T_STORAGE_R_NV(t0, SV_UDSI_G(phi1))))
        beta0=0;  /* if gradient is not allocated and stored yet,
                     bypass the following macro (it happens
                     when case/data files are being read */
      else
        BOUNDARY_SECONDARY_GRADIENT_SOURCE(beta0, SV_UDSI_G(phi1),dG,
           es, A_by_es, gamma);

  /* temporary variables used in the profile expression */
      temp1=gamma*A_by_es/dr;
      temp2=hC*Af;


      F_PROFILE(f, thread, nv)
        = (temp1*C_UDSI(c0, t0, phi1)- beta0 + temp2*PHI1_inf)/(temp2 + temp1);
    }
  end_f_loop(f, thread)

    }
}
```

Appendix B: Unsteady UDS: 1st-Order Implicit

Note: Add the program fragment to the code listed in Appendix A before compiling the source code. The complete UDF is in `transientMixedBC.c`.

```
/************************************************************************/
/* Implementation of                                                    */
/*                                                                      */
/*  the unsteady term for a user-defined scalar (1st-order)             */
/*                                                                      */
/************************************************************************/


DEFINE_UDS_UNSTEADY(unst1stOrder, c, t, i, apu, su)
{
  /* if the unsteady term is different from the default term: d(rho*phi)
                                                             ----------
                                                                dt
     this macro is used to specify the appropriate unstready term      */

  real volume, cp=CP, deltaTime=CURRENT_TIMESTEP;

  volume=C_VOLUME(c, t);

  /* the transient term is moved to the RHS of the equation and is split
    into two parts---check the FVM algorithm for detail. First-order
    backward differencing is implemented below:                       */

  *apu = -cp*volume/deltaTime;
  *su  =  cp*volume*C_UDSI_M1(c, t, i)/deltaTime;

}
```