Miscellaneous Functions/Macros

Advanced UDF Modeling Course

Trigonometric Functions

`double acos (double x);`	returns the arc-cosine of x
`double asin (double x);`	returns the arc-sine of x
`double atan (double x);`	returns the arc-tangent of x
`double atan2 (double x, double y);`	returns the arc-tangent of x/y
`double cos (double x);`	returns the cosine of x
`double sin (double x);`	returns the sine of x
`double tan (double x);`	returns the tangent of x
`double cosh (double x);`	returns the hyperbolic cosine of x
`double sinh (double x);`	returns the hyperbolic sine of x
`double tanh (double x);`	returns the hyperbolic tangent of x

Miscellaneous Math-Functions

◆ `double sqrt (double x);`	returns the square root of x
◆ `double pow (double x, double y);`	returns x^y
◆ `double exp (double x);`	returns e^x
◆ `double log (double x);`	returns ln(x)
◆ `double log10 (double x);`	returns $\ln_{10}(x)$
◆ `double fabs (double x);`	returns |x|
◆ `double ceil (double x);`	smallest integer not less than x
◆ `double floor (double x);`	largest integer not greater than x

- The macro **`UNIVERSAL_GAS_CONSTANT`** returns the value of the universal gas constant (8314.34), which is expressed in SI units of J/Kmol-K
- The macro **`M_PI`** returns the value of π

Standard I/O Functions

- use **`Message`** instead of **`printf`** in compiled UDFs (UNIX only)

```
Message ("Volume integral: %g\n", sum_vol);
```

◆ **`FILE *fopen`(char *filename, char *type);**	opens a file
◆ **`int fclose`(FILE *fd);**	closes a file
◆ **`int fprintf`(FILE *fd, char *format, ...);**	formatted print to a file
◆ **`int printf`(char, *format, ...);**	print to screen
◆ **`int fscanf`(FILE *fd, char *format, ...);**	formatted read from a file

See your system manual pages for more details

Note that for parallel runs, the I/O macros need to be different

- Example:

```
FILE *fd;
real f1, f2;
fd = fopen(“data.txt”,”r”);
fscanf(fd, “%f %f”,&f1,&f2);
fclose(fd);
```

Special Macro’s

`cxboolean Data_Valid_P()`	Equals `1` if data is available, `0` if not Usage: `if(!Data_Valid_P())return;`
`cxboolean FLUID_THREAD_P(t0)`	true if thread t0 fluid thread
`cxboolean SOLID_THREAD_P(t0)`	true if thread t0 is solid thread
`cxboolean BOUNDARY_FACE_THREAD_P(t0)`	true if thread t0 is boundary thread
`NULLP(T_STORAGE_R_NV(t0, SV_UDSI_G(p1)))`	- Checks for storage allocation of user defined scalars
`CURRENT_TIME`	Real current flow time (in seconds)
`CURRENT_TIMESTEP`	Real current physical time step size (in sec)
`PREVIOUS_TIME`	Real previous flow time (in seconds)
`PREVIOUS_2_TIME`	Real flow time two steps back in time (in sec)
`N_TIME`	Integer number of time steps
`N_ITER`	Integer number of iterations

Miscellaneous: Vector Utilities

- `ND_ND` in the declaration of a vector or matrix stands for the actual fluent dimension (2D / 3D)
- `X[ND_ND]` is equivalent to:
  - 2D: `X[2]`
  - 3D: `X[3]`
- `NV_MAG` computes the magnitude of a vector: `X[ND_ND]`
- `NV_MAG(x)` is equivalent to:
  - 2D: `sqrt(x[0]*x[0] + x[1]*x[1]);`
  - 3D: `sqrt(x[0]*x[0] + x[1]*x[1] + x[2]*x[2]);`
- `NV_MAG2` computes the sum of squares of vector components
- `NV_MAG2(x)` is equivalent to:
  - 2D: `(x[0]*x[0] + x[1]*x[1]);`
  - 3D: `(x[0]*x[0] + x[1]*x[1] + x[2]*x[2]);`

Miscellaneous: Vector Utilities

- `ND_SUM` computes the sum of `ND_ND` arguments
- `ND_SUM(x,y,z)` is equivalent to:
  - 2D: `x + y;`
  - 3D: `x + y + z;`
- `ND_SET` generates `ND_ND` assignment statements
  - 2D: `ND_SET(u,v,C_U(c,t),C_V(c,t))` is equivalent to:
    - `u = C_U(c, t);`
    - `v = C_V(c, t);`
  - 3D: `ND_SET(u,v,w,C_U(c,t),C_V(c,t),C_W(c,t))`is equivalent to:
    - `u = C_U(c, t);`
    - `v = C_V(c, t);`
    - `w = C_W(c, t);`

Miscellaneous: Vector Utilities

- `NV_V` performs an operation on two vectors
  - `NV_V(a, =, x);`
  - `a[0] = x[0]; a[1] = x[1];` etc.
  - Note that if you use `+ =` instead of `=` in the above equation, then you get `a[0]+= x[0];` etc.
- `NV_VV` is a vector operator . The operation that is performed on the elements depends upon what is used as an argument in place of the + signs
  - `NV_VV(a, =, x, +, y)`/* The '+' symbol can be replaced by (-, /,*) */
  - 2D: `a[0]= x[0]+y[0], a[1]= x[1]+y[1];`
  - 3D: `a[0]= x[0]+y[0], a[1]= x[1]+y[1], a[2]= x[2]+y[2];`

Miscellaneous: Vector Utilities

- **`NV_V_VS`** adds a vector to another which is multiplied by a scalar
  - **`NV_V_VS(a,=, x,+,y,*,0.5);`**
  - 2D: **`a[0]=x[0]+(y[0]*0.5),  a[1]=x[1]+(y[1]*0.5);`**
  - Note that + sign can be replaced by -, /, or *, and '*' sign can be replaced by '/'
- **`NV_VS_VS`** adds a vector to another which are each multiplied by a scalar
  - **`NV_VS_VS(a,=,x,*,2.0,+,y,*,0.5);`**
  - 2D: **`a[0]=(x[0]*2.0)+(y[0]*0.5),`**
    `a[1]=(x[1]*2.0)+(y[1]*0.5);`
  - Note that + sign can be used in place of -, *, or /, and '*' sign can be replaced by '/'

Miscellaneous: Vector Utilities

- The dot products of two sets of vector or components
- **`ND_DOT(x,y,z,u,v,w)`** is equivalent to:
  - **`2D: (x*u+y*v);`**
  - **`3D: (x*u+y*v+z*w);`**
- **`NV_DOT(x,u)`** is equivalent to:
  - **`2D: (x[0]*u[0]+x[1]*u[1]);`**
  - **`3D: (x[0]*u[0]+x[1]*u[1]+x[2]*u[2]);`**
- **`NVD_DOT(x,u,v,w)`** is equivalent to:
  - **`2D: (x[0]*u+x[1]*v);`**
  - **`3D: (x[0]*u+x[1]*v+x[2]*w);`**
- **`NV_CROSS(a,x,y)`** is available for **`3D`** only:
  - It returns the cross product of vectors **x** and **y** in the new vector **a**

Closure

- All UDF-s must be written in SI units
- UDF-s open up a virtually endless opportunity to extend the modeling capabilities of the basic FLUENT code
- Details of the examples and all working macros & parameters are available in the UDF manual at Fluent User Services Center