How to access solid material properties of a porous equilibrium zone in an ANSYS Fluent User-Defined Function (UDF)?

Description

For the thermal equilibrium porous model Fluent doesn't create an additional zone for the porous solid. Therefore, you cannot access material properties with the usual `C_*` macros. If you need access to the material properties of the solid zone, you can access the data structure directly. It is only available for compiled UDFs and cannot be used when interpreting.

Important

Whenever you access the data structure, keep in mind that this can change without prior notice. Changes will not be documented in the release notes or the migration manual. If your UDF relies on macros that are not documented in the ANSYS Fluent Customization Manual, ANSYS can decline providing technical support. If you observe unusual behavior, reproduce the behavior without using undocumented macros before contacting the ANSYS technical support.

Solution

The Fluent data structure for a porous zone includes the material properties of the solid for the equilibrium thermal model. But there is no macro to access them.

The material properties can be constant, temperature-dependent or user-defined. The different temperature-dependent methods are not stored. For example, if you first define your specific heat capacity as piecewise-linear and then define it as piecewise-polynomial, the first definition is overwritten. However, the constant value is always kept, even if you use a temperature-dependent formulation.

This is important because you need to access the correct material property from your UDF.

To access a property directly from the data structure you can use `THREAD_SOLID_MATERIAL` or `THREAD_MATERIAL`, depending on which part of the structure you want to access. For the solid material of an equilibrium porous zone you must use `THREAD_SOLID_MATERIAL`. Both are defined in `threads.h`.

To access a property, you use the arrow operator. Solid materials have only three properties available: density, specific heat capacity and thermal conductivity.

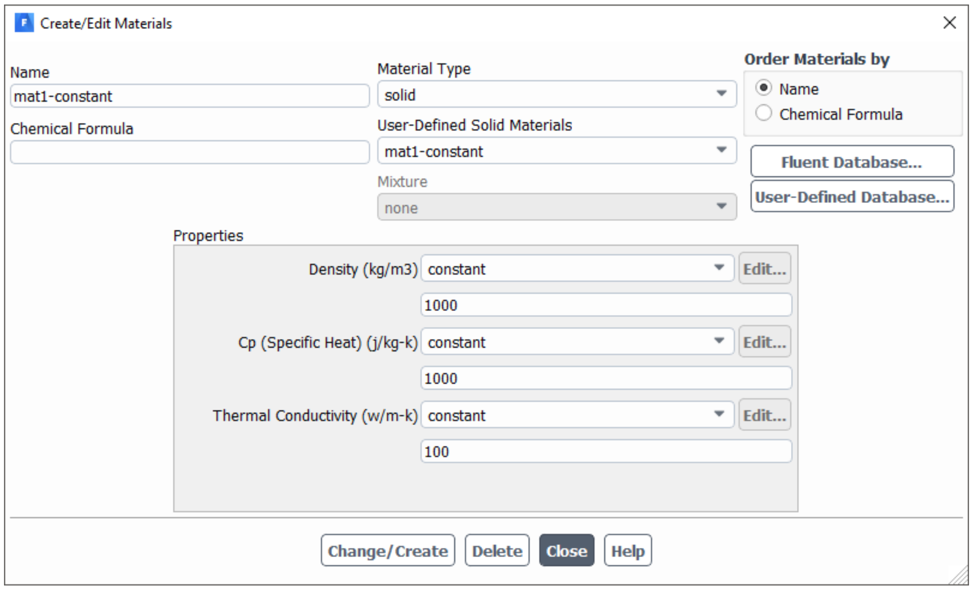
```
THREAD_SOLID_MATERIAL(t)->p[PROP_rho]
THREAD_SOLID_MATERIAL(t)->p[PROP_Cp]
THREAD_SOLID_MATERIAL(t)->p[PROP_ktc]
```

`t` is used for the cell thread pointer.

`PROP_ktc` is only valid for isotropic thermal conductivity. For anisotropic thermal conductivity, you can access `PROP_ktc0`, `PROP_ktc1` and `PROP_ktc2`. These are *not discussed* in this document. It can be necessary to use additional macros to get the correct thermal conductivity for anisotropic behavior.

As mentioned earlier, you can access constant and temperature-dependent values. To get the constant value, use the dot operator to grab it directly:

```
THREAD_SOLID_MATERIAL(t)->[PROP_Cp].constant
```



Name: mat1-constant
Chemical Formula:
Material Type: solid
User-Defined Solid Materials: mat1-constant
Mixture: none
Order Materials by: [✓ Name] Chemical Formula

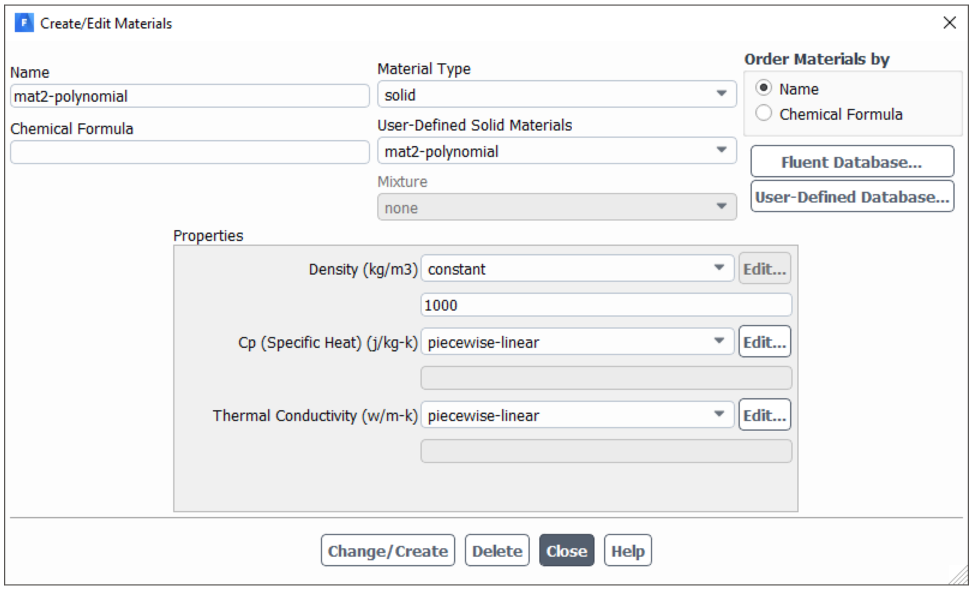
Properties

Density (kg/m3)	constant	1000
Cp (Specific Heat) (j/kg-k)	constant	1000
Thermal Conductivity (w/m-k)	constant	100

To access the temperature-dependent value, you must use another macro to calculate it:

```
MATERIAL_PROP_POLYNOMIAL(THREAD_SOLID_MATERIAL(t),PROP_Cp,C_T(c,t))
```

Again, `t` is used for the cell thread pointer and `c` for the cell index cell_t.
Although the macro has ‘polynomial’ in its name, it returns the correct value for all possible specifications.



Name: mat2-polynomial
Chemical Formula:
Material Type: solid
User-Defined Solid Materials: mat2-polynomial
Mixture: none
Order Materials by: [✓ Name] Chemical Formula

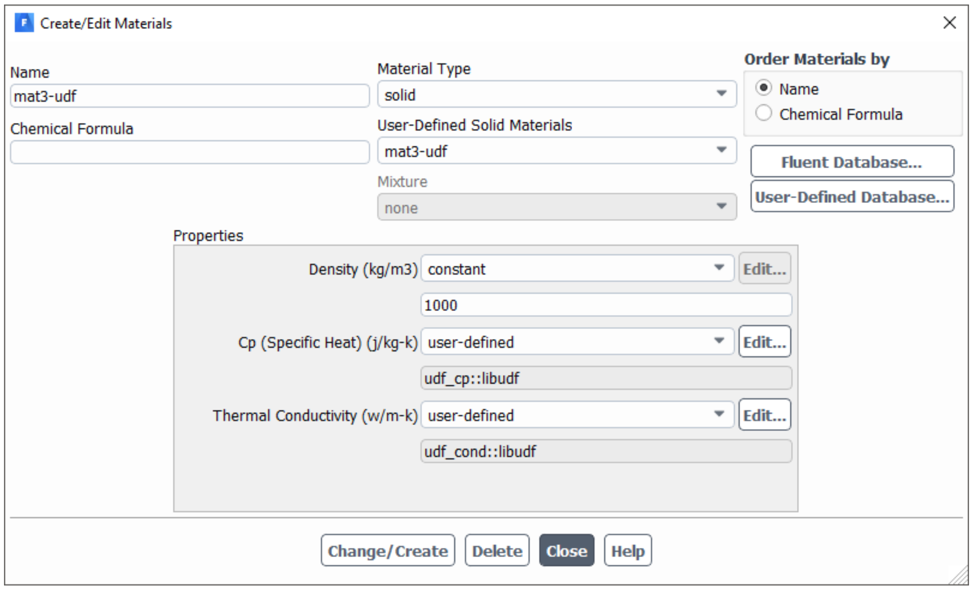
Properties

Density (kg/m3)	constant	1000
Cp (Specific Heat) (j/kg-k)	piecewise-linear	
Thermal Conductivity (w/m-k)	piecewise-linear	

So far, you can access the two values regardless of which one of them is defined in the case. To use the correct one, you can check which method is used with the dot operator:

```
THREAD_SOLID_MATERIAL(t)->p[PROP_Cp].method
```

This is `0` when constant is active, `1` for any of the temperature-dependent methods and `2` when you have a UDF hooked for that property.



Create/Edit Materials

Name	mat3-udf
Chemical Formula	
Material Type	solid
User-Defined Solid Materials	mat3-udf
Mixture	none
Order Materials by	[✓ Name] Chemical Formula

Properties

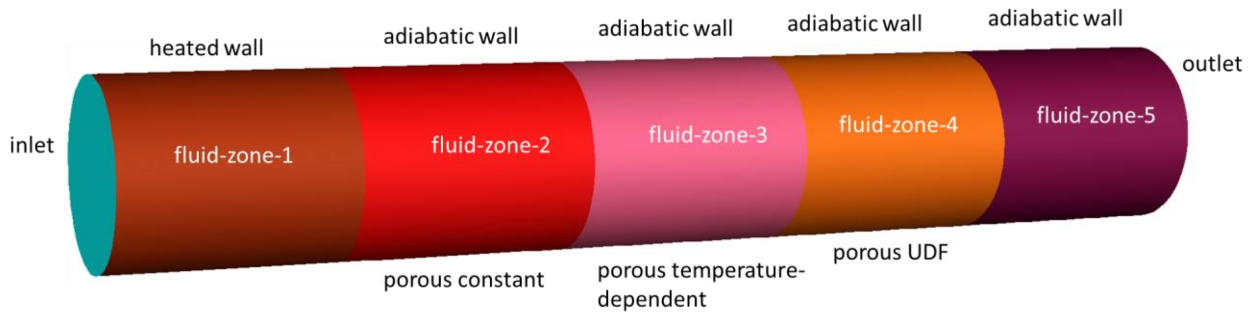
Density (kg/m3)	constant	1000
Cp (Specific Heat) (j/kg-k)	user-defined	udf_cp::libudf
Thermal Conductivity (w/m-k)	user-defined	udf_cond::libudf

It is not possible to also grab the value of a UDF through the data structure. Instead, you should define the calculation of the property in its own function that you can call it separately.

Example

It is often easier to understand the usage of UDF macros with an example. The demonstration case is a simple straight pipe that has five cell zones. The default air with constant material properties is used. The first wall is heated with a fixed wall temperature of 500K while the inlet temperature is 300K. The remaining walls are adiabatic.

The second to fourth cell zones are defined as porous zones with three different solid materials. Fluid-zone-2 has a constant material hooked, fluid-zone-3 a material with piecewise-linear definition and fluid-zone-4 a UDF with a comparable definition as zone 3. There is no additional resistance defined in the zones.



The source code consists of two parts. The first part defines the material properties for the specific heat and the thermal conductivity for zone 4.

The second part is an ON_DEMAND function that reports the minimum, maximum and average values of the used solid material.

To use the example, compile the UDF as `libudf`, read the case and run the calculation. Once converged, execute the ON_DEMAD `check_porous_properties` to print the results into the Fluent console.

```c
001  #include "udf.h"
002
003  real calc_thermal_conductivity(real temperature)
004  {
005    return 100.0 + temperature/5.0;
006  }
007
008  real calc_specific_heat(real temperature)
009  {
010    return 1000.0 + temperature*2.0;
011  }
012
013  DEFINE_SPECIFIC_HEAT(udf_cp, T, Tref, h, yi)
014  {
015    real cp = calc_specific_heat(T);
016    *h = cp * (T-Tref);
017    return cp;
018  }
019
020  DEFINE_PROPERTY(udf_cond, c, t)
021  {
022    return calc_thermal_conductivity(C_T(c, t));
023  }
```

Line 1:	Only `udf.h` is required to include
Line 3-6:	Calculation of the thermal conductivity as a separate function. This makes it easier to access it later in the code. It requires the temperature as input parameter
Line 8-11:	Calculation of the specific heat capacity as a separate function. It requires the temperature as input parameter
Line 13-18:	Definition of the specific heat that can be hooked in the Fluent materials panel. It calls the function defined earlier and calculates the sensible enthalpy, as required by Fluent
Line 20-23:	Definition of the thermal conductivity that can be hooked in the Fluent materials panel. It just calls the function with the cell temperature

The rest of the code is quite long and discussed in pieces, that the explanations are closer to the relevant segments of the code.

```
025 /* Calculation of min, max and av values from within a cell loop */
026 void min_max_av(real *value, real *max, real *min, real *av, real *volume,
    real *vol_sum)
027 {
028   if (*value > *max) {
029     *max = *value;
030   }
031   if (*value < *min) {
032     *min = *value;
033   }
034   *vol_sum += *volume;
035   *av += *value * *volume;
036 }
```

Line 26:	Function to calculate min, max and average values inside a cell loop. This is needed several times and should operate on different variables. Therefore, pointers are used as parameters
Line 28-30:	Replace the max value if the current value is larger than the old maximum
Line 31-33:	Replace the min value if the current value is smaller than the old minimum
Line 34-35:	First part of the averaging. The rest must be done after the global reduction outside of the cell loop

```
038 DEFINE_ON_DEMAND(check_porous_properties)
039 {
040 #if !RP_HOST
041   Domain *d = Get_Domain(1);
042   Thread *t;
043   cell_t c;
044   real min_cp, max_cp, av_cp, cp, cp_sum;
045   real min_tc, max_tc, av_tc, tc, tc_sum;
046   real density;
047   real volume, volume_sum;
```

Line 38:	Start of the ON_DEMAND function
Line 40:	The whole UDF should only be executed on compute nodes. Nothing is done on the host process
Line 41-47:	Variable declarations for domain, thread, cell index and the material properties

```
049   thread_loop_c(t, d)
050   {
051     min_cp = 1000000.0;
052     max_cp = 0.0;
053     av_cp  = 0.0;
054     min_tc = 1000000.0;
055     max_tc = 0.0;
056     av_tc  = 0.0;
057     cp_sum = 0.0;
058     tc_sum = 0.0;
059     volume_sum = 0.0;
060
061     if (POROUS_THREAD_P(t)) {
062       density = THREAD_SOLID_MATERIAL(t)->p[PROP_rho].constant;
```

Line 49:	Start the loop over all cell threads that exist. The loop ends at the end of the function
Line 51-59:	Initialize the variables on each compute node. Min values are large, max values are small. This makes it easy to find the local and global extreme values
Line 61:	Check if the current cell zone is a porous zone. This if condition also ends at the end of the thread loop
Line 62:	Get the density. For solid materials there is only the constant material specification. Although it is possible to define this value with a UDF, too, this is not commonly done. If you want to make the UDF cover that case, you need to implement a similar condition as it is used for the other properties.

```
064       /* Specific heat capacity */
065       if (THREAD_SOLID_MATERIAL(t)->p[PROP_Cp].method == 0) {
066         /* cp is constant */
067         min_cp = THREAD_SOLID_MATERIAL(t)->p[PROP_Cp].constant;
068         max_cp = min_cp;
069         av_cp = min_cp;
070         volume_sum = 1.0;
```

Line 65-70:	Execute only if specific heat is defined as constant
Line 67-69:	Get the specific heat capacity and store it in min, max and average variables
Line 70:	Set the volume to 1 for the calculation of the average value later on. This saves a couple of lines of code to treat the output differently for constant and variable material properties

```
071     } else if (THREAD_SOLID_MATERIAL(t)->p[PROP_Cp].method == 1) {
072       /* temperature-dependent specification of cp */
073       begin_c_loop_int(c, t)
074       {
075         cp = MATERIAL_PROP_POLYNOMIAL(THREAD_SOLID_MATERIAL(t),
                                          PROP_Cp,C_T(c, t));
076         volume = C_VOLUME(c,t);
077         min_max_av(&cp, &max_cp, &min_cp, &av_cp, &volume, &volume_sum);
078       }
079       end_c_loop_int(c, t)
```

Line 71-79:	Execute only if specific heat is defined as temperature-dependent directly in the materials panel
Line 73-79:	Loop over all cells that are native to the current compute node
Line 75:	Calculate the correct specific heat from the definition in the materials panel and the cell temperature
Line 76:	Get the cell volume for calculating the average
Line 77:	Call the function `min_max_av` with the references to the different variables to decide if the specific heat of the current cell is a new maximum or minimum and to build the sum for averaging

```
080     } else if (THREAD_SOLID_MATERIAL(t)->p[PROP_Cp].method == 2) {
801       /* UDF specification of cp */
082       begin_c_loop_int(c, t)
083       {
084         cp = calc_specific_heat(C_T(c,t));
085         volume = C_VOLUME(c,t);
086         min_max_av(&cp, &max_cp, &min_cp, &av_cp, &volume, &volume_sum);
087       }
088       end_c_loop_int(c, t)
089     } else {
090       Message0("Error, access to solid specific heat not possible\n");
091       return;
092     }
```

Line 80-88:	Execute only if specific heat is defined by a UDF
Line 82-88:	Loop over all cells that are native to the current compute node
Line 84:	Calculate the specific heat for the temperature of the current cell by calling the same function that is also used in the `DEFINE_SPECIFIC_HEAT` function
Line 85-86:	Call the function to determine min, max and average values
Line 89-92:	If the method is not `0`, `1` or `2`, report an error and end the UDF. This should never be called if the material specification is valid

```
094        max_cp = PRF_GRHIGH1(max_cp);
095        min_cp = PRF_GRLOW1(min_cp);
096        PRF_GRSUM2(av_cp, volume_sum);
097        av_cp /= volume_sum;
098        volume_sum = 0.0;
```

Line 94: Find the global maximum over all compute nodes and synchronize the variable on all nodes

Line 95: Find the global minimum over all compute nodes and synchronize the variable on all nodes

Line 96: Calculate the global sum over all compute nodes for the nominator and the denominator to calculate the average value

Line 97: Calculate the volume and mass average for the specific heat capacity (density is constant, volume and mass average are identical)

Line 98: Reset the volume to `0` for the following calculations

```
100        /* Thermal conductivity */
101        if (THREAD_SOLID_MATERIAL(t)->p[PROP_ktc].method == 0) {
102          /* thermal conductivity is constant */
103          min_tc = THREAD_SOLID_MATERIAL(t)->p[PROP_ktc].constant;
104          max_tc = min_tc;
105          av_tc = min_tc;
106          volume_sum = 1.0;
107        } else if (THREAD_SOLID_MATERIAL(t)->p[PROP_ktc].method == 1) {
108          /* temperature-dependent specification of thermal conductivity */
109          begin_c_loop_int(c, t)
110          {
111            tc = MATERIAL_PROP_POLYNOMIAL(THREAD_SOLID_MATERIAL(t),
                                             PROP_ktc,C_T(c, t));
112            volume = C_VOLUME(c,t);
113            min_max_av(&tc, &max_tc, &min_tc, &av_tc, &volume, &volume_sum);
114          }
115          end_c_loop_int(c, t)
116        } else if (THREAD_SOLID_MATERIAL(t)->p[PROP_ktc].method == 2) {
117          /* UDF specification of thermal conductivity */
118          begin_c_loop_int(c, t)
119          {
120            tc = calc_thermal_conductivity(C_T(c,t));
121            volume = C_VOLUME(c,t);
122            min_max_av(&tc, &max_tc, &min_tc, &av_tc, &volume, &volume_sum);
123          }
124          end_c_loop_int(c, t)
125        } else {
126          Message0("Error, access to solid thermal conductivity not
possible\n");
127          return;
128        }
129
130        max_tc = PRF_GRHIGH1(max_tc);
131        min_tc = PRF_GRLOW1(min_tc);
132        PRF_GRSUM2(av_tc, volume_sum);
133        av_tc /= volume_sum;
```

Line 100-133: Repeat the procedure for the thermal conductivity

```
135           Message0("Zone %d\n\n", THREAD_ID(t));
136           Message0("Specific heat capacity\n");
137           Message0("Max: %e    | Min: %e    | Average: %e\n", max_cp, min_cp,
                                                                  av_cp);
138           Message0("Thermal conductivity\n");
139           Message0("Max: %e    | Min: %e    | Average: %e\n", max_tc, min_tc,
                                                                  av_tc);
140           Message0("Density\n");
141           Message0("Constant value: %e\n", density);
142           Message0("\n\n");
143         }
144       }
145     #endif
146     }
```

Line 135-142: Output of the calculated values
Line 143: End the if-condition that checks for porous threads
Line 144: End the loop over all cell threads
Line 145: End the preprocessor directive to execute the code only on the compute nodes
Line 146: End the ON_DEMAND UDF

Attachments

1. 2059991_demonstration.zip – Demonstration case with complete UDF

Keywords: ANSYS Fluent; UDF; user-defined function; user defined functions; material; property; properties; solid; porous zone; specific heat capacity; density; thermal conductivity; access; read; constant; temperature-dependent; temperature dependent; user-specified; user specified; THREAD_SOLID_MATERIAL; POROUS_THREAD_P; MATERIAL_PROP_POLYNOMIAL; data structure

Contributors: Akram Radwan