ALTAIR

ONLY FORWARD

Altair® AcuSolve® 2025

AcuTrace User-Defined Functions Manual

Updated: 11/13/2024

Contents

AcuTrace User-Defined Functions Manual

1

Instructions to define additional solution quantities of AcuTrace called user equations.

AcuTrace is a particle tracer that runs as a post-processor to or a co-processor with AcuSolve. AcuTrace computes particle traces as a series of segments using a fifth-order time-discontinuous Galerkin (TDG) method with error control for solving ordinary differential equations. AcuTrace typically solves for particle position and, optionally, particle stretch.

The user equations are governed either by evolution equations of the form

$$\frac{D\vec{s}_p}{Dt} = \vec{f}(\vec{s}_p, \vec{x}_p, t, \vec{l}_p, \vec{S}_p, \vec{U}_f; \vec{V}_{udf}) \quad (1)$$

or by evaluation equations of the form

$$\vec{s}_p = \vec{f}(\vec{s}_p, \vec{x}_p, t, \vec{l}_p, \vec{S}_p, \vec{U}_f; \vec{V}_{udf}) \quad (2)$$

where $\vec{s}_p$ is the set of user defined particle variables in the current equation, $\vec{S}_p$ is the set of user defined variables from the other user equations (if any), $\vec{U}_f$ is the set of flow variables, and $\vec{V}_{udf}$ is the set of user equation parameters. User-defined functions are used to define the right hand side of these additional evolution and evaluation equations.

Two steps are needed to use user-defined functions:

- Build a user-defined function
- Reference it in the input file

User-Defined Functions in AcuTrace are referred to as UFPs (for User-Defined Function Particle) to differentiate them from the UDFs used in AcuSolve.

Any number of user-defined functions may be used in a given problem. As further explained below, the user defined functions are compiled and linked into one or more dynamic shared libraries. The script acuMakeLib on Unix/Linux machines and acuMakeDll on Windows machines may be used for this purpose. A list of these libraries is then given to AcuTrace via the configuration option user_libraries. The solver then sequentially searches through these libraries for the user-defined functions referenced in the input file.

Input File

2

To access a user-defined function in AcuTrace, you must reference it in the input file.

For example, to access the user-defined function **`usrEner()`**, a user equation needs to be defined and then added to the list of user equations. To define the user equation energy, the following command may be specified in the input file:

```
USER_EQUATION( "energy" ) {
     num_variables = 1
     user_values   = {5.0, 0.015}
     user_string   = {"test"}
     type          = evolve
}
```

This user equation command defines a user equation with one component governed by an evolution equation. The right hand side of the evolution equation is found by evaluating **`usrEner`**. The value of the *`user_function`* parameter is exactly the name of the function (routine) used in the C program written by you. The parameters *`user_values`* and *`user_strings`* are used to pass parameters to the user function. In this example, two floating-point numbers and one string are passed to the function. The order of these parameters is preserved, for example the first user value is 5.0 and the second is 0.015.

The user equations defined in `USER_EQUATION` commands can be added to the list of user equations in the `EQUATION` command. For example,

```
particle       = massless
user_equations = {energy}
}
```

A user equation that has been defined and added to the list of user equations is part of the particle solution field. If a user equation is defined but not referenced in the `EQUATION` command, it is not part of the particle solution field and has no effect. The initial values of the user equation come from one or more `USER_EQUATION_INITIAL_CONDITION` commands, for example,

```
USER_EQUATION_INITIAL_CONDITION (energy) {
     particle_seed   = seeds
     user_equation   = energy
     type            = constant
     constant_values = { 10.0 }
}
```

> **Note:** The value of the *`user_equation`* parameter in the `USER_EQUATION_INITIAL_CONDITION` command is the qualifier used in the `USER_EQUATION` command, not the name of the C function.

A user equation in the list of user equations in the `EQUATION` command will automatically be advanced if the `AUTO_SOLUTION_STRATEGY` is used, for example,

```
AUTO_SOLUTION_STRATEGY {
     max_time     = 0.0
```

```
      max_segments = 10000
}
```

Otherwise, the user equation is advanced only if the user equation is referenced in a `STAGGER` command that is in turn referenced by the `TIME_SEQUENCE` command, for example,

```
TIME_SEQUENCE {
      ...
      staggers = {particle, energy}
      ...
}
STAGGER(energy) {
      equation      = user_equation
      user_equation = energy
}
```

If the user equation is not so referenced, it simply retains its initial values throughout the analysis. See the AcuTrace Command Reference Manual for more details on the inputs for user-defined functions.

Function Format

3

An AcuTrace user-defined function has the following form (in C).

```
#include "acusim.h"
#include "ufp.h"

UFP_PROTOTYPE(usrFunc);

Void usrFunc (
      UfpHd     ufpHd,          /* Opaque handle for accessing data  */
      Real*     outVec,         /* Output Vector                     */
      Integer   nItems,         /* Number of items in outVec         */
      Integer   vecDim,         /* Vector dimension of outVec        */
)
{
      ...
      outVec[0] = ... ;
      ...
} /*  end of usrFunc() */
```

Note:

- The order of the arguments is slightly different than the order in an AcuSolve user-defined function.
- The first argument is "*ufpHd*" not "*udfHd*".
- The header file "`ufp.h`" is included, not "`udf.h`".

The header file "`acusim.h`" defines the standard C header file, such as "`stdio.h`" and "`stdarg.h`". In addition, the data types used by AcuTrace are defined. The relevant types are:

Integer	type integer
Real	type floating point
String	type string
Void	type void

The header file "`ufp.h`" contains the definitions and declarations needed by the user function:

- The definitions of symbolic constants, such as `UFP_PARTICLE_MARKER` and `UFP_PARTICLE_ID`. These constants are used to access data.
- The prototypes of support routines
- The macro `UFP_PROTOTYPE()`, which may be used to prototype the user function.

In addition, it contains the useful macros **`min(a,b)`**, **`max(a,b)`** and **`abs(a)`** for the computation of minimum, maximum, and absolute value of integers or floating point numbers.

The user-function name (generically called **`usrFunc`** above) may be any name supported in C that starts with the letters **`usr`**. This naming convention avoids any potential conflict in function name space.

Four arguments are passed to a user-defined function:

ufpHd This is an opaque handle (pointer) which contains the necessary information for accessing various data. All supporting routines require this argument. For example, to access the current time pass *ufpHd* to the function **ufpGetTime()**, as in:

```
Real time ;
      ...
      time = ufpGetTime( ufpHd ) ;
```

outVec This is the resulting vector of the user function. You must fill this vector before returning. On input this array contains all zeros.

nItems This is the first dimension of *outVec*, which specifies the number of items that needs to be filled.

vecDim This is the second dimension of *outVec*, which specifies the vector dimension of the particular data.

The following example computes the source term in an evolution equation for particle energy:

```
#include "acusim.h"
#include "ufp.h"

UFP_PROTOTYPE(usrEnergy);
Void usrEnergy (
         UfpHd   ufpHd,       /* Opaque handle for accessing data */
         Real*   outVec       /* Output Vector                    */
         Integer nItems,      /* Number of items in outVec        */
         Integer vecDim,      /* Vector dimension of outVec       */
)
{
         Real*   usrVals ;    /* user values                */
         Real*   t_fluid ;    /* fluid temperature          */
         Real*   h_particle ; /* particle energy            */
         Real    mpXcp ;      /* mass * c_p for particle    */
         Real    cond ;       /* conductivity               */
         Real*   jac ;        /* source jacobian            */
      usrVals    = ufpGetUsrVals( ufpHd ) ;
      cond       = usrVals[0] ;
      mpXcp      = usrVals[1] ;
      t_fluid    = ufpGetFlowData( ufpHd,UFP_FLOW_TEMPERATURE ) ;
      h_particle = ufpGetUdfData( ufpHd, 0 ) ;
      outVec[0]  = -cond * ( h_particle[0] / mpXcp - t_fluid [0] ) ;

      jac        = ufpGetJac( ufpHd, UFP_JAC_UDF_VARIABLES ) ;
      jac[0]     = -cond / mpXcp ;
}
```

AcuTrace only accesses user-defined functions that are written in C. To write a user-function in Fortran, you must write a C cover function via which Fortran routine(s) are called. This cover function is accessed by AcuTrace. For the above example the C file could be rewritten as:

```
#include "acusim.h"
#include "ufp.h"
```

```
UFP_PROTOTYPE(usrEnergy);

Void usrEnergy (
      UfpHd   ufpHd,       /* Opaque handle for accessing data */
      Real*   outVec,      /* Vector dimension of outVec       */
      Integer nItems,      /* Number of items in outVec        */
      Integer vecDim,      /* Vector dimension of outVec       */
)
{
      Real*   usrVals ;    /* user values                      */
      Real*   t_fluid ;    /* fluid temperature                */
      Real*   h_particle ; /* particle energy                  */
      Real*   jac ;        /* source jacobian                  */
      usrVals     = ufpGetUsrVals( ufpHd ) ;
      t_fluid     = ufpGetFlowData( ufpHd, UFP_FLOW_TEMPERATURE ) ;
      h_particle  = ufpGetUdfData( ufpHd, 0 ) ;
      jac         = ufpGetJac( ufpHd, UFP_JAC_UDF_VARIABLES     ) ;

      usrEnergyF( usrVals, t_fluid, h_particle, jac, outVec ) ;
}
```

while the Fortran file may contain:

```
        subroutine usrEnergyF ( usrVals, t_fluid, h_particle, jac, outVec )
        real*8 usrVals(2), t_fluid, h_particle, jac, outVec
        real*8 cond, mpXcp

          cond       = usrVals(1)
          mpXcp      = usrVals(2)
          outVec       = -cond * ( h_particle / mpXcp - tFluid )
          jac      = -cond / mpXcp

          return
          end
```

Note: You are responsible for Fortran/C name conversion.

The fastest dimension of *outVec* is *nItems*, and not *vecDim*. For example, to fill *outVec* with the flow velocity at the location of particle, you can write the following code:

```
#include "acusim.h"
#include "ufp.h"

UFP_PROTOTYPE(usrVel);

Void usrVel (
      UfpHd   ufpHd,             /* Opaque handle for accessing data     */
      Real*   outVec,            /* Output Vector                        */
      Integer nItems,            /* Number of items in outVec            */
      Integer vecDim             /* Vector dimension of outVec           */
)

{
      Real *data ;
      Integer dir, item ;
      if ( vecDim != 3 ) {
          printf( "vecDim = %d, but must equal 3 instead\n", vecDim ) ;
          exit( 0 ) ;
      }
      data = ufpGetFlowData( ufpHd, UFP_FLOW_VELOCITY ) ;
```

```
        for ( dir = 0 ; dir < 3 ; dir++ ) {
            for ( item = 0 ; item < nItems; item++ ) {
                outVec[item+dir*nItems] = data[dir] ;
            }
        }
}
```

In the current uses of AcuTrace user defined functions, *nItems* always equals 1.

Compile, Link and Run

4

In order for AcuTrace to access the user-defined functions, the functions must be compiled and linked into a shared library. The script acuMakeLib may be used for this purpose.

Assume you have the files "`usrEner.c`" and "`usrEnerF.f`" containing the C and Fortran functions given in the previous section. To compile and link into the shared library `libusr.so` simply issue the command:

```
acuMakeLib -src usrEner.c,usrEnerF.f
```

This script writes the makefile, `Makefile`, and invokes make to compile and link the routines. The makefile has two targets, make all, or equivalently make install, and make clean.

Once the file is successfully compiled and linked, the particle tracer may be invoked as

```
acuRunTrace -libs ./libusr.so
```

Multiple user-defined functions may be provided by one or more libraries. To run the particle tracer with multiple libraries, give a comma separated list of libraries to the configuration option -libs. For example, given the libraries `libener1.so` and `libener2.so` stored in directory `~/acusim_libs`, the particle tracer may be invoked as

```
acuRunTrace -libs ~/acusim_libs/libener1.so,~/acusim_libs/libener2.so
```

The libraries are searched sequentially in the order given for the user functions. Libraries that do not exist are ignored. To see what libraries and what routines are accessed by the solver, invoke acuRunTrace with the option -verbose 2.

Note: For most computer platforms, the functions within a given user shared library can access all functions in that library, plus all system and solver functions. However, they may not access functions in other user shared libraries. This behavior is a platform dependent.

Note: On most platforms the shared library has a `.so` extension, however on some it may be `.sl`, such as HP, or `.dll`, Intel/WIN.

UFP-Support Routines

5

Within an AcuTrace user-defined function, data may be accessed through support routines. These routines provide flexibility, longevity, efficiency and the ability to check for run-time errors.

The support routines fall into these categories:

- Basic Routines: General UFP routines, such as user values and setting error flags.
- Particle Routines: Access particle data, such as position and stretch vector.
- Flow Routines: Access flow data, such as fluid velocity and pressure.

Basic Routines

6

These routines are accessible by all AcuTrace user-defined functions.

This chapter covers the following:

ufpGetType()

Return the type of the user-defined function.

Syntax

```
type = ufpGetType ( ufpHd ) ;
```

Type

AcuSolve User-Defined Function Basic Routine

Parameters

ufpHd (pointer)
The opaque handle which was passed to the user function.

Return Value

The type of the user-defined function is returned as an integer.

Description

This routine returns the type of a user-defined function. There is currently no type differentiation among AcuTrace user defined functions; the function always returns a value of 1. This routine is reserved for future use. For example,

```
Integer type ;
...
type = ufpGetType( ufpHd ) ;
```

Errors

This routine expects a valid *ufpHd* as an argument; an invalid argument returns an error.

ufpGetName()

Return the user-given name of the input command associated with the user function.

Syntax

```
name = ufpGetName ( ufpHd ) ;
```

Type

AcuSolve User-Defined Function Basic Routine

Parameters

ufpHd (pointer)
: The opaque handle which was passed to the user function.

Return Value

The returned string value is the user-given name of the input command calling this user function.

Description

This routine returns the user-given name of the input command associated with the user function. This facilitates correlating with the input file. For example,

```
String user_name ;
...
user_name = ufpGetName( ufpHd ) ;
printf( "Inside command with name %s\n", user_name ) ;
```

Errors

This routine expects a valid *ufpHd* as an argument; an invalid argument returns an error.

ufpCheckNumUsrVals()

Check the number of user values given in the input file.

Syntax

ufpCheckNumUsrVals (ufpHd, nUsrVals) ;

Type

AcuSolve User-Defined Function Basic Routine

Parameters

ufpHd (pointer)
: The opaque handle which was passed to the user function.

nUsrVals (integer)
: Number of user values that the function expects.

Return Value

None

Description

This routine verifies that the number of user values supplied in the input file is the same as the second argument. If it is not, an error message is issued and the particle tracer exits. For example,

```
ufpCheckNumUsrVals( ufpHd, 2 ) ;
```

Errors

This routine expects a valid *ufpHd*.

ufpGetNumUsrVals()

Return the number of user values supplied in the input file.

Syntax

```
nUsrVals = ufpGetNumUsrVals ( ufpHd ) ;
```

Type

AcuSolve User-Defined Function Basic Routine

Parameters

ufpHd (pointer)
The opaque handle which was passed to the user function.

Return Value

The number of user values given in the input file is returned as an integer.

Description

This routine returns the number of user values supplied in the input file. It is useful for determining that the correct number of parameters was supplied. It is also useful when the number of parameters is not know a priori. For example,

```
Integer nUsrVals ;
Real x, y ;
Real usrVals[2] ;
nUsrVals = ufpGetNumUsrVals( ufpHd ) ;
if ( nUsrVals != 2 ) {
   printf( "nUsrVals does not equal 2\n" ) ;
   exit( 0 ) ;
}
usrVals  = ufpGetUsrVals( ufpHd ) ;
x = usrVals[0] ;
y = usrVals[1] ;
```

Errors

This routine expects a valid *ufpHd*.

ufpGetUsrVals()

Return the array of user-supplied values.

Syntax

```
usrVals = ufpGetUsrVals ( ufpHd ) ;
```

Type

AcuSolve User-Defined Function Basic Routine

Parameters

ufpHd (pointer)
The opaque handle which was passed to the user function.

Return Value

The return value is a pointer to one-dimensional real array of user values as given in the input file. This array has **ufpGetNumUsrVals()** entries.

Description

This routine returns the real array of user-given parameters. The order of the parameters within the array is the same as in the input file. For example,

```
Real* usrVals ;
Real amp, omega ;
...
ufpCheckNumUsrVals( ufpHd, 2 ) ;
usrVals = ufpGetUsrVals( ufpHd ) ;
amp     = usrVals[0] ;
omega   = usrVals[1] ;
```

Errors

This routine expects a valid *ufpHd*.

ufpCheckNumUsrStrs()

Check the number of user strings given in the input file.

Syntax

ufpCheckNumUsrStrs (ufpHd, nUsrStrs) ;

Type

AcuSolve User-Defined Function Basic Routine

Parameters

ufpHd (pointer)
: The opaque handle which was passed to the user function.

nUsrStrs (integer)
: Number of user strings that the function expects.

Return Value

None

Description

This routine verifies that the number of user strings supplied in the input file is the same as the second argument. If it is not, an error message is issued and the particle tracer exits. For example,

```
ufpCheckNumUsrStrs( ufpHd, 2 ) ;
```

Errors

This routine expects a valid *ufpHd*.

ufpGetNumUsrStrs()

Return the number of user strings supplied in the input file.

Syntax

nUsrStrs = ufpGetNumUsrStrs (ufpHd) ;

Type

AcuSolve User-Defined Function Basic Routine

Parameters

ufpHd (pointer)
The opaque handle which was passed to the user function.

Return Value

The return value is the number of user strings given in the input file.

Description

This routine returns the number of user strings supplied in the input file. For example,

```
Integer nUsrStrs ;
...
nUsrStrs = ufpGetNumStrs( ufpHd ) ;
if ( nUsrStrs != 2 ) {
   printf( "Invalid number of user strings %d\n", nUsrStrs ) ;
   exit(0);
}
```

Errors

This routine expects a valid *ufpHd*.

ufpGetUsrStrs()

Return the array of user supplied strings.

Syntax

usrStrs = ufpGetUsrStrs (ufpHd) ;

Type

AcuSolve User-Defined Function Basic Routine

Parameters

ufpHd (pointer)
The opaque handle which was passed to the user function.

Return Value

The return value is an array of user strings given in the input file. This array has `ufpGetNumUsrStrs()` entries.

Description

This routine returns the array of user strings given in the input file. For example,

```
String*      usrStr ;
usrStr       = ufpGetUsrStrs( ufpHd ) ;
/*-----------------------------------------------------------------------
* Get the solution for user equation usrStr[0]
* -----------------------------------------------------------------------
*/
ufpVal       = ufpGetUfpData( ufpHd, usrStr[0] ) ;
```

Errors

This routine expects a valid *ufpHd*.

Particle Routines

7

These routines are accessible by all AcuTrace user-defined functions. The routines access the particle data at the current particle location and time.

This chapter covers the following:

ufpGetTime()

Get the particle time.

Syntax

```
time = ufpGetTime ( ufpHd ) ;
```

Type

AcuTrace User-Defined Function Particle Routine

Parameters

`ufpHd` (pointer)
The opaque handle which was passed to the user function.

Return Value

The return value is the particle time (real value). This time is not the time returned by the AcuSolve User-Defined Function `udfGetTime`.

Description

This routine returns the current particle time. For example,

```
Real  time;
...
time = ufpGetTime( ufpHd ) ;
```

Errors

This routine expects a valid *`ufpHd`* as an argument; an invalid arguments returns an error.

ufpGetTimeInc()

Get the time increment for the current particle segment.

Syntax

```
dt = ufpGetTimeInc ( ufpHd ) ;
```

Type

AcuTrace User-Defined Function Particle Routine

Parameters

`ufpHd` (pointer)
: The opaque handle which was passed to the user function

Return Value

The return value is the particle time increment (real value). This time increment is not the time increment returned by the AcuSolve User-Defined Function `udfGetTimeInc`.

Description

This routine returns the time increment for the current particle segment. For example,

```
Real  dt;
...
dt = ufpGetTimeInc( ufpHd ) ;
```

Errors

This routine expects a valid *`ufpHd`* as an argument; an invalid argument returns an error.

ufpGetParticleData()

Get the array of particle data.

Syntax

```
data = ufpGetParticleData( ufpHd, dataName ) ;
```

Type

AcuTrace User-Defined Function Particle Routine

Parameters

ufpHd (pointer)
The opaque handle which was passed to the user function.

dataName (integer)
Symbolic name of the requested data.

Table 1:

Symbolic Name	Data returned	Array size
UFP_PARTICLE_ID	particle id	1
UFP_PARTICLE_MARKER	marker	1
UFP_PARTICLE_POSITION	position	3
UFP_PARTICLE_VELOCITY	velocity	3
UFP_PARTICLE_LENGTH	stretch vector	3
UFP_PARTICLE_MASS	mass	1
UFP_PARTICLE_DENSITY	density	1
UFP_PARTICLE_RADIUS	radius	1
UFP_PARTICLE_VOLUME	volume	1

Return Value

The return value is a pointer to a one-dimensional real array.

The size of the array depends on the value of *dataName*.

Particle user defined equation data is not accessed by this routine but instead by ufpGetUdfData.

Description

This routine returns the array of particle data at the current particle location and time. For example,

```
      Real *pCrd ;
...
      pCrd = ufpGetParticleData( ufpHd, UFP_PARTICLE_POSITION ) ;

/*-----------------------------------------------------------------------------
* Compute source term for residence time:
* Set source to 1 if the particle is in the residence box, 0 otherwise
*-----------------------------------------------------------------------------
*/

if  ( pCrd[0] >= xlo && pCrd[0] <= xhi &&
      pCrd[1] >= ylo && pCrd[1] <= yhi &&
      pCrd[2] >= zlo && pCrd[2] <= zhi ) {
      outVec[0]       = 1.0 ;

} else {
      outVec[0]       = 0.0 ;
}
```

Errors

This routine expects valid *ufpHd* and *dataName* as arguments; invalid arguments return an error.

ufpGetJac()

Get the pointer to the source Jacobian.

Syntax

```
jac = ufpGetJac ( ufpHd ) ;
```

Type

AcuTrace User-Defined Function Particle Routine

Parameters

ufpHd (pointer)
The opaque handle which was passed to the user function.

Return Value

The return value is a pointer to a one-dimensional array. Once the pointer is obtained, you should fill it with the values of the Jacobian with respect to the user function variables. The size of the Jacobian is square of the number of user equation variables.

If the source term does not depend on the user function variable, the Jacobian is identically 0 and does not need to be set.

It is your responsibility to determine the correct formulation of the Jacobian.

Description

This routine returns the pointer to the source Jacobian.

A user defined function providing the source term for heat transfer between the fluid and the particle could be written as:

```
Void usrTemp (
      UfpHd   ufpHd,
      Real*   outVec,
      Integer nItems,
      Integer vecDim
)
{
      Real* jac ;
      Real* usrVals ;
      Real* t_fluid ;
      Real* t_particle ;
      Real  cond ;
...
      usrVals           = ufpGetUsrVals (  ufpHd                                ) ;
      jac               = ufpGetJac     (  ufpHd, UFP_JAC_UDF_VARIABLES         ) ;
      t_fluid           = ufpGetFlowData(  ufpHd,UFP_FLOW_TEMPERATURE           ) ;
      t_particle        = ufpGetUdfData (  ufpHd,0                              ) ;
      cond              = usrVals[0];
      outVec[0)         = cond * ( t_fluid[0] - t_particle[0]

 /*-----------------------------------------------------------------------
* The jacobian is the derivative of the source with respect to the particle
```

```
* temperature
*-------------------------------------------------------------------------
*/
      jac[0]            = -cond ;
...
}
```

In this case, the evolution equation is

$$\frac{DT_p}{Dt} = S = k(T_f - T_p) \tag{3}$$

where T_p is the particle temperature, T_f the fluid temperature, and k the conductivity. The Jacobian is

$$\frac{\partial S}{\partial T_p} = -k. \tag{4}$$

Errors

This routine expects a valid `ufpHd` as an argument; an invalid arguments returns an error.

ufpGetUdfData()

Get the pointer to the user equation solution.

Syntax

data = ufpGetUdfData (ufpHd, eqnName) ;

Type

AcuTrace User-Defined Function Particle Routine

Parameters

ufpHd (pointer)
: The opaque handle which was passed to the user function.

eqnName (string)
: The name of the user equation. If the integer 0 is used instead of a string, the current user equation is used.

Return Value

The return value is a pointer to a one dimensional array containing the solution for the user equation eqnName.

eqnName should correspond to the qualifier used in the USER_EQUATION command in the trace input file. If the integer 0 is used instead of a string, the solution vector of the current user equation is returned. The example in the previous section illustrates this usage.

Description

This routine returns the pointer to the user equation solution at the current particle position and time.

In this example there is one user equation for particle energy and a second one for particle temperature. The temperature is given by $T_p = \frac{e_p}{c_p}$ where T_p is the particle temperature, e_p is the particle energy, and c_p is the particle specific heat.

The definitions of these two equations in the trace input file could be be written as

```
USER_EQUATION( "energy" ) {
      user_function              = "usrEnergy"
      num_variables              = 1
   ...
}
USER_EQUATION( "temp" ) {
      user_function              = "usrTemp"
      num_variables              = 1
      type                       = evaluate
      user_values                = {C_P}
}

usrTemp could be written as

UFP_PROTOTYPE(usrTemp);
```

```
Void usrTemp (
      UfpHd   ufpHd,
      Real*   outVec,
      Integer nItems,
      Integer vecDim
)
{
      Real*      e_particle ;
      Real*      c_p ;
      Real*      usrVals ;
      usrVals                  = ufpGetUsrVals( ufpHd ) ;
      c_p                      = usrVals[0] ;
      e_particle               = ufpGetUdfData( ufpHd, "energy" ) ;
      outVec[0]                = e_particle[0] / cp_p ;
}
```

Errors

This routine expects valid *ufpHd* and *eqnName* as arguments; invalid arguments return an error.

ufpGetNumUdfData()

Get the number of values in the user equation solution.

Syntax

```
n = ufpGetNumUdfData ( ufpHd, eqnName ) ;
```

Type

AcuTrace User-Defined Function Particle Routine

Parameters

ufpHd (pointer)
: The opaque handle which was passed to the user function.

eqnName (string)
: The name of the user equation. If the integer 0 is used instead of a string, the current user equation is used.

Return Value

The return value is the number of values (integer) in the solution for the user equation *eqnName*.

eqnName should correspond to the qualifier used in the USER_EQUATION command in the trace input file. If the integer 0 is used instead of a string, the number of values in the solution of the current user equation is returned.

Description

This routine returns the size of the user equation solution.

In this example there is a user equation with three values in its solution vector. The definition of this equation in the trace input file could be be written as

```
USER_EQUATION( "example" ) {
      user_function      = "usrExample"
      num_variables      = 3
...
}
```

num_vars would then be set to 3 in the following code segment:

```
Integer            num_vars ;
num_vars           = ufpGetNumUdfData( ufpHd, "example" ) ;
```

If the code segment is in usrExample,

```
num_vars           = ufpGetNumUdfData( ufpHd, 0) ;
```

would also set *num_vars* to 3.

Errors

This routine expects valid *ufpHd* and *eqnName* as arguments; invalid arguments return an error.

Flow Routines

8

These routines are accessible by all AcuTrace user-defined functions. The routines access the flow data at the current particle location and time.

This chapter covers the following:

ufpGetFlowData()

Get the array of flow data.

Syntax

```
data = ufpGetFlowData ( ufpHd, dataName ) ;
```

Type

AcuSolve User-Defined Function Flow Routine.

Parameters

ufpHd (pointer)
The opaque handle which was passed to the user function.

dataName (integer)
Symbolic name of the requested data.

Table 2:

Symbolic Name	Data returned	Array size
UFP_FLOW_VELOCITY	flow velocity	3
UFP_FLOW_PRESSURE	pressure	1
UFP_FLOW_TEMPERATURE	temperature	1
UFP_FLOW_EDDY_VISCOSITY	eddy viscosity	1
UFP_FLOW_SPECIES_1	species 1	1
UFP_FLOW_SPECIES_2	species 2	1
UFP_FLOW_SPECIES_3	species 3	1
UFP_FLOW_SPECIES_4	species 4	1
UFP_FLOW_SPECIES_5	species 5	1
UFP_FLOW_SPECIES_6	species 6	1
UFP_FLOW_SPECIES_7	species 7	1
UFP_FLOW_SPECIES_8	species 8	1
UFP_FLOW_SPECIES_9	species 9	1
UFP_FLOW_GRAD_VELOCITY	flow velocity gradient	9

Symbolic Name	Data returned	Array size
UFP_FLOW_GRAD_PRESSURE	pressure gradient	3
UFP_FLOW_GRAD_TEMPERATURE	temperature gradient	3
UFP_FLOW_GRAD_EDDY_VISCOSITY	eddy viscosity gradient	3
UFP_FLOW_GRAD_SPECIES_1	species 1 gradient	3
UFP_FLOW_GRAD_SPECIES_2	species 2 gradient	3
UFP_FLOW_GRAD_SPECIES_3	species 3 gradient	3
UFP_FLOW_GRAD_SPECIES_4	species 4 gradient	3
UFP_FLOW_GRAD_SPECIES_5	species 5 gradient	3
UFP_FLOW_GRAD_SPECIES_6	species 6 gradient	3
UFP_FLOW_GRAD_SPECIES_7	species 7 gradient	3
UFP_FLOW_GRAD_SPECIES_8	species 8 gradient	3
UFP_FLOW_GRAD_SPECIES_9	species 9 gradient	3

Return Value

The return value is a pointer to a one-dimensional real array.

The size of the array depends on the value of *dataName*. You are responsible for ensuring that the requested data is available in the AcuSolve data base.

Description

This routine returns the array of flow data at the current particle location and time. For example,

```
Real*     data ;
Real      ext ;
data      = ufpGetFlowData( ufpHd, UFP_FLOW_GRAD_VELOCITY ) ;
ext       = sqrt(data[0]*data[0] + data[4]*data[4] + data[8]*data[8]) ;
```

Errors

This routine expects valid *ufpHd* and *dataName* as arguments; invalid arguments return an error.

ufpGetExtData()

Get the array of extended flow data.

Syntax

data = ufpGetExtData (ufpHd, dataName) ;

Type

AcuSolve User-Defined Function Flow Routine.

Parameters

ufpHd (pointer)
The opaque handle which was passed to the user function.

dataName (integer)
Name of the requested data.

Return Value

The return value is a pointer to a one dimensional real array.

The size of the array depends on the value of *dataName*. You are responsible for ensuring that the requested data is available in the AcuSolve database. *dataName* must be specified in the *extended_flow_variables* parameter of the FLOW_FIELD command in the trace input file.

Description

This routine returns the array of extended flow data at the current particle location and time.

To access *material_viscosity* in a user defined function, *material_viscosity* must first be specified in the *extended_flow_variables* parameter in the trace input file:

```
FLOW_FIELD {
...
      extended_flow_variables      =
                                   {
                                   material_viscosity,
                                   ...
                                   }
...
}
```

In the C source file, the code may be written as:

```
Real*                data;
Real                 material_viscosity ;
data                 = ufpGetExtData( ufpHd, "material_viscosity" ) ;
material_viscosity   = data[0] ;
```

Errors

This routine expects valid *ufpHd* and *dataName* as arguments; invalid arguments return an error.

ufpGetExtNVars()

Get the size of the extended flow data.

Syntax

```
nVars = ufpGetExtNVars ( ufpHd, dataName ) ;
```

Type

AcuSolve User-Defined Function Flow Routine.

Parameters

ufpHd (pointer)
: The opaque handle which was passed to the user function.

dataName (integer)
: Name of the requested data.

Return Value

The return value is the size of the one dimensional real array that `ufpGetExtData` (*ufpHd*, *dataName*) would return.

You are responsible for ensuring that the requested data is available in the AcuSolve data base. *dataName* must be specified in the *extended_flow_variables* parameter of the `FLOW_FIELD` command in the trace input file.

This function is useful primarily if you do not know the size of the array returned by `ufpGetExtData` from before.

Description

This routine returns the size of the extended flow data.

In the trace input file, you could have

```
FLOW_FIELD {
...
      extended_flow_variables      =
                                   {
                                   grad_viscosity,
                                   ...
                                   }
...
}
```

In a user defined function, the lines

```
Integer size ;
size        = ufpGetExtNVars( ufpHd, "grad_velocity" ) ;
```

would set the value of size to 9.

Errors

This routine expects valid *ufpHd* and *dataName* as arguments; invalid arguments return an error.

Intellectual Property Rights Notice

This Intellectual Property Rights Notice is exemplary, and therefore not exhaustive, of the intellectual property rights held by Altair Engineering Inc. or its affiliates. Software, other products, and materials of Altair Engineering Inc. or its affiliates are protected under laws of the United States and laws of other jurisdictions.

In addition to intellectual property rights indicated herein, such software, other products, and materials of Altair Engineering Inc. or its affiliates may be further protected by patents, additional copyrights, additional trademarks, trade secrets, and additional other intellectual property rights. For avoidance of doubt, copyright notice does not imply publication. Copyrights in the below are held by Altair Engineering Inc. or its affiliates. Additionally, all non-Altair marks are the property of their respective owners. If you have any questions regarding trademarks or registrations, please contact marketing and legal.

This Intellectual Property Rights Notice does not give you any right to any product, such as software, or underlying intellectual property rights of Altair Engineering Inc. or its affiliates. Usage, for example, of software of Altair Engineering Inc. or its affiliates is governed by and dependent on a valid license agreement.

Altair® HyperWorks®, a Design & Simulation Platform

Altair® AcuSolve® ©1997-2024

Altair® Activate® ©1989-2024

Altair® Automated Reporting Director™ ©2008-2022

Altair® Battery Damage Identifier™ ©2019-2024

Altair® CFD™ ©1990-2024

Altair Compose® ©2007-2024

Altair® ConnectMe™ ©2014-2024

Altair® DesignAI™ ©2022-2024

Altair® DSim™ ©2024

Altair® EDEM™ ©2005-2024

Altair® EEvision™ ©2018-2024

Altair® ElectroFlo™ ©1992-2024

Altair Embed® ©1989-2024

Altair Embed® SE ©1989-2024

Altair Embed®/Digital Power Designer ©2012-2024

Altair Embed®/eDrives ©2012-2024

Altair Embed® Viewer©1996-2024

Altair® e-Motor Director™©2019-2024

Altair® ESAComp® ©1992-2024

Altair® expertAI™ ©2020-2024

Altair® Feko® ©1999-2024

Altair® FlightStream® ©2017-2024

Altair® Flow Simulator™ ©2016-2024

Altair® Flux® ©1983-2024

Altair® FluxMotor® ©2017-2024

Altair® GateVision PRO™ ©2002-2024

Altair® Geomechanics Director™ ©2011-2022

Altair® HyperCrash® ©2001-2023

Altair® HyperGraph® ©1995-2024

Altair® HyperLife® ©1990-2024

Altair® HyperMesh® ©1990-2024

Altair® HyperMesh® CFD ©1990-2024

Altair® HyperMesh ® NVH ©1990-2024

Altair® HyperSpice™ ©2017-2024

Altair® HyperStudy® ©1999-2024

Altair® HyperView® ©1999-2024

Altair® HyperView Player® ©2022-2024

Altair® HyperWorks® ©1990-2024

Altair® HyperWorks® Design Explorer ©1990-2024

Altair® HyperXtrude® ©1999-2024

Altair® Impact Simulation Director™ ©2010-2022

Altair® Inspire™ ©2009-2024

Altair® Inspire™ Cast ©2011-2024

Altair® Inspire™ Extrude Metal ©1996-2024

Altair® Inspire™ Extrude Polymer ©1996-2024

Altair® Inspire™ Form ©1998-2024

Altair® Inspire™ Mold ©2009-2024

Altair® Inspire™ PolyFoam ©2009-2024

Altair® Inspire™ Print3D ©2021-2024

Altair® Inspire™ Render ©1993-2024

Altair® Inspire™ Studio ©1993-2024

Altair® Material Data Center™ ©2019-2024

Altair® Material Modeler™ ©2019-2024

Altair® Model Mesher Director™ ©2010-2024

Altair® MotionSolve® ©2002-2024

Altair® MotionView® ©1993-2024

Altair® Multi-Disciplinary Optimization Director™ ©2012-2024

Altair® Multiscale Designer® ©2011-2024

Altair® newFASANT™ ©2010-2020

Altair® nanoFluidX® ©2013-2024

Altair® NLVIEW® ©2018-2024

Altair® NVH Director™ ©2010-2024

Altair® NVH Full Vehicle™ ©2022-2024

Altair® NVH Standard™ ©2022-2024

Altair® OmniV™ ©2015-2024

Altair® OptiStruct® ©1996-2024

Altair® physicsAI™ ©2021-2024

Altair® PollEx™ ©2003-2024

Altair® PollEx™ for ECAD ©2003-2024

Altair® PSIM™ ©1994-2024

Altair® Pulse™ ©2020-2024

Altair® Radioss® ©1986-2024

Altair® romAI™ ©2022-2024

Altair® RTLvision PRO™ ©2002-2024

Altair® S-CALC™ ©1995-2024

Altair® S-CONCRETE™ ©1995-2024

Altair® S-FRAME® ©1995-2024

Altair® S-FOUNDATION™ ©1995-2024

Altair® S-LINE™ ©1995-2024

Altair® S-PAD™ ©1995-2024

Altair® S-STEEL™ ©1995-2024

Altair® S-TIMBER™ ©1995-2024

Altair® S-VIEW™ ©1995-2024

Altair® SEAM® ©1985-2024

Altair® shapeAI™ ©2021-2024

Altair® signalAI™ ©2020-2024

Altair® Silicon Debug Tools™ ©2018-2024

Altair® SimLab® ©2004-2024

Altair® SimLab® ST ©2019-2024

Altair® SimSolid® ©2015-2024

Altair® SpiceVision PRO™ ©2002-2024

Altair® Squeak and Rattle Director™ ©2012-2024

Altair® StarVision PRO™ ©2002-2024

Altair® Structural Office™ ©2022-2024

Altair® Sulis™ ©2018-2024

Altair® Twin Activate® ©1989-2024

Altair® UDE™ ©2015-2024

Altair® ultraFluidX® ©2010-2024

Altair® Virtual Gauge Director™ ©2012-2024

Altair® Virtual Wind Tunnel™ ©2012-2024

Altair® Weight Analytics™ ©2013-2022

Altair® Weld Certification Director™ ©2014-2024

Altair® WinProp™ ©2000-2024

Altair® WRAP™ ©1998-2024

Altair® HPCWorks®, a HPC & Cloud Platform

Altair® Allocator™ ©1995-2024

Altair® Access™ ©2008-2024

Altair® Accelerator™ ©1995-2024

Altair® Accelerator™ Plus ©1995-2024

Altair® Breeze™ ©2022-2024

Altair® Cassini™ ©2015-2024

Altair® Control™ ©2008-2024

Altair® Desktop Software Usage Analytics™ (DSUA) ©2022-2024

Altair® FlowTracer™ ©1995-2024

Altair® Grid Engine® ©2001, 2011-2024

Altair® InsightPro™ ©2023-2024

Altair® Hero™ ©1995-2024

Altair® Liquid Scheduling™ ©2023-2024

Altair® Mistral™ ©2022-2024

Altair® Monitor™ ©1995-2024

Altair® NavOps® ©2022-2024

Altair® PBS Professional® ©1994-2024

Altair® PBS Works™ ©2022-2024

Altair® Simulation Cloud Suite (SCS) ©2024

Altair® Software Asset Optimization (SAO) ©2007-2024

Altair® Unlimited™ ©2022-2024

Altair® Unlimited Data Analytics Appliance™ ©2022-2024

Altair® Unlimited Virtual Appliance™ ©2022-2024

Altair® RapidMiner®, a Data Analytics & AI Platform

Altair® AI Hub ©2023-2024

Altair® AI Edge™ ©2023-2024

Altair® AI Cloud ©2022-2024

Altair® AI Studio ©2023-2024

Altair® Analytics Workbench™ ©2002-2024

Altair® Graph Lakehouse™ ©2013-2024

Altair® Graph Studio™ ©2007-2024

Altair® Knowledge Hub™ ©2017-2024

Altair® Knowledge Studio® ©1994-2024

Altair® Knowledge Studio® for Apache Spark ©1994-2024

Altair® Knowledge Seeker™ ©1994-2024

Altair® IoT Studio™ ©2002-2024

Altair® Monarch® ©1996-2024

Altair® Monarch® Classic ©1996-2024

Altair® Monarch® Complete™ ©1996-2024

Altair® Monarch® Data Prep Studio ©2015-2024

Altair® Monarch Server™ ©1996-2024

Altair® Panopticon™ ©2004-2024

Altair® Panopticon™ BI ©2011-2024

Altair® SLC™ ©2002-2024

Altair® SLC Hub™ ©2002-2024

Altair® SmartWorks™ ©2002-2024

Altair® RapidMiner® ©2001-2024

Altair One® ©1994-2024

Altair® CoPilot™ ©2023-2024

Altair® License Utility™ ©2010-2024

Altair® TheaRender® ©2010-2024

OpenMatrix™ ©2007-2024

OpenPBS® ©1994-2024

OpenRadioss™ ©1986-2024

October 7, 2024

Technical Support

Altair's support resources include engaging learning materials, vibrant community forums, intuitive online help resources, how-to guides, and a user-friendly support portal.

Visit Customer Support to learn more about our support offerings.

Index