Exploring Journals with Gambit and Fluent

A. Mukhopadhyay
am@fluent.com

Objectives…

- Ease of use during
  - Parametric study
  - Repeat study with changes in process parameters
  - Minor design changes
  - Scale up
  - Sensitivity analysis
- Think of setting up a complicated matrix of case studies and accomplish the study by not-so-savvy helping hands
  - Large scale CFD deployment
- Improvise equipments/processes or practices based on systematic analysis
  - Automate most components of the analysis procedure to maximize analysis productivity and reduce probability of user error
- Embed CFD on front-end of your choice
  - Fluent / gambit / Excel
- Journals are very effective tools for CFD record keeping

Outline

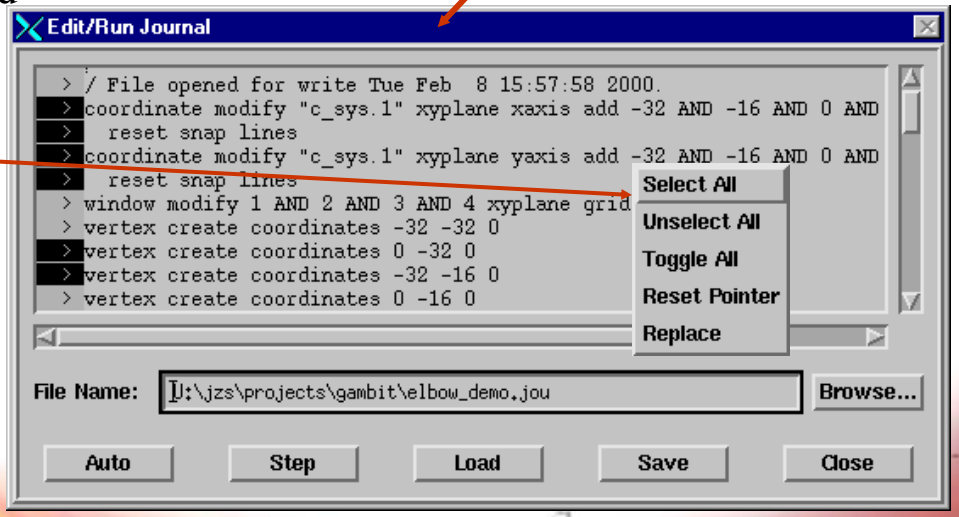
- Journaling in Gambit
  - Basis of Journal Files
  - Parameters: Scalars and Arrays
  - Special Constants
  - Expressions: Arithmetic, Logical and String
  - Functions: String and Arithmetic
  - DO and IF-THEN-ELSE Commands
  - Steps to parameterize GAMBIT Journals
- Journaling in Fluent
  - Text User Interface (TUI)
  - Parameterize variables in TUI
  - Create your own GUI in Fluent
  - Run Gambit journal in the back ground
  - Post-processing and html reports through journals
  - Easy post-processing of previously stored transient data
- Excel to drive gambit and fluent

Journal Files

- Journal File:
  - Executable list of Gambit commands
    - Created automatically by Gambit from GUI and TUI
    - Can be edited or created externally with text editor
  - Journals are small - easy to transfer, e-mail, store
- Uses:
  - Can be parameterized, comments can be added
  - Easy recovery from a crash or power loss
  - Edit existing commands to create new ones
  - Information about a model can be found without the need to run GAMBIT
  - Bugs can easily be reproduced - faster fixing

Running Journal Files

- Journal files can be processed in two ways:
  - Batch mode (Run)
    - All commands processed without interruption
    - "read pause" command will force interrupt with resume option appearing
  - Interactive mode (Edit/Run)
    - Includes text editor for easy modifications
      - Mark lines in process field to activate for processing
      - Editable text field
      - Right click text field for more options
      - Auto or Step through activated process lines



Journal File: Parametric Modeling

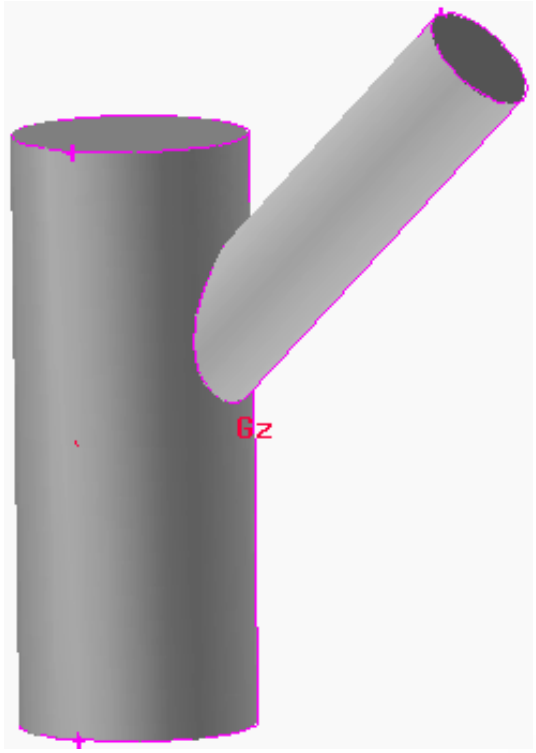
- Parameters (including arrays), control-blocks, do-loops, arithmetic functions, etc., can be used in the Journal File for simplifying parametric studies.

```
> /parameters
> /large cylinder height ($lh) and radius ($lr)
> $lh=20.
> $lr=4.
> /small cylinder height ($sh) and radius ($sr)
> $sh=18.
> $sr=2.
> $ang=45.
> /end parameters
> /
> undo begingroup
> volume create height $l                3 $lr xaxis frustum
> undo endgroup
> /
> undo begingroup
> volume create height $sh radius1 $sr radius3 $sr offset 0 0 10 zaxis frustum
> undo endgroup
> /
> /rotate small cylinder
> /
> undo begingroup
> volume move "volume.2" dangle $ang vector 0 1 0 origin 0 0 0
> undo endgroup
> /
> /unite cylinders
> /
> volume unite "volume.1" volume "volume.2"
> /
```

Parameter names begin with $. Parameters are case sensitive.

GAMBIT Commands are not case sensitive

Comment lines



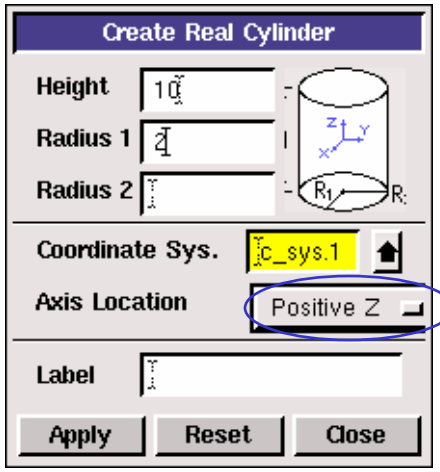
Command Interpreter (1)

- Commands are not case sensitive
- Comments begin with **/**
  - **`/ This is a comment line`**
- Continue statements with ****
  - **`vertex create coordinates \`**
    `0.0 1.0 2.0`
- Special commands
  - **`reset`** (deletes all entities)
  - **`reset mesh`** (deletes all mesh)
  - **`read file "`***filename*`"`** (reads a journal file)
  - **`read pause`** (pauses journal when using the **Run** (not **Run/Edit**) option; click **Resume** button to continue)
- All commands and arguments are documented in *GAMBIT Command Reference Guide*

Parameters

- Scalar or Array
- Numeric or string
- Defined by: $param = *value*
  - param = name of parameter
  - *value* = numeric or string value of parameter
- Name of parameter
  - Must start with $
  - Is not case sensitive (**`$length`** same as **`$LENGTH`**)

Scalar: Pipe



Cylinder: Height = 10, Radius = 2

Axis Location: Positive Z

Center of the cylinder is offset (Height / 2) in the + z direction from the origin of the active coordinate system

offsets in the x, y and z directions

Positive Z

```
/original journal file
volume create height 10 radius1 2 radius3 2 offset 0 0 0 zaxis frustum
volume create height 10 radius1 2 radius3 2 offset 0 0 5 zaxis frustum
```

Centered Z

```
/modified journal file with parameters for height ($h) & radius ($r)
$h = 10
$r = 2
volume create height $h radius1 $r radius3 $r offset 0 0 ($h/2) zaxis frustum
```

Use Parenthesis

Array (1)

- Define arrays by **`declare $p[{n1}:m1, {n2}:m2, ...]`**
  - Where **`p`** is the name of the parameter
  - **`n`** is the starting index ({} indicate this is optional; default is 1)
  - **`m`** is the range of the dimension
  - Square brackets [] are necessary
- Elements in the array still need to have values assigned to them
  - **`$p[1,2]= 6.5`**
- **`declare $sides[4]`** Creates
  `$sides[1], $sides[2], $sides[3], $sides[4]`
- **`declare $tri[2:3]`** Creates
  `$tri[2], $tri[3], $tri[4]`
- **`declare $sqr[3, 2]`** Creates
  `$sqr[1,1], $sqr[1,2], $sqr[2,1] $sqr[2,2], $sqr[3,1], $sqr[3,2]`
- **`declare $matrix[0:3, 5:2]`** Creates
  `$matrix[0,5], $matrix[0,6]`
  `$matrix[1,5], $matrix[1,6]`
  `$matrix[2,5], $matrix[2,6]`

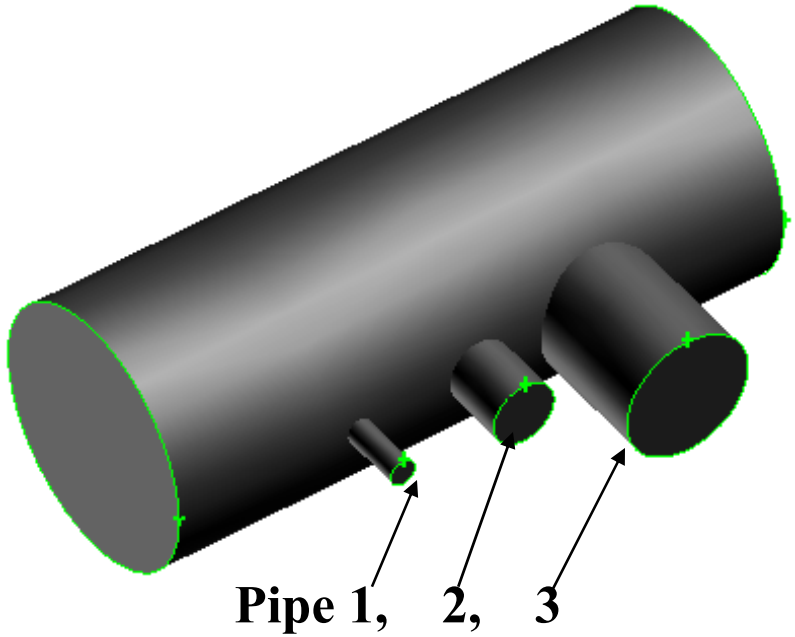
Array (2): Multiple Pipes

```
declare $p[3,2]
```

1st dimension is the pipe number (1, 2 or 3)

2nd dimension is the radius (1) or height (2)

Pipe #	Radius	Height
1	`$p[1,1]=.5`	`$p[1,2]=3`
2	`$p[2,1]=1`	`$p[2,2]=3`
3	`$p[3,1]=2`	`$p[3,2]=4`



- Globally available constants:
  - `PI` 3.141592653590
  - `TWOPI` 6.283185307180
  - `DEG2RAD` 0.0174532925199
  - `RAD2DEG` 57.29577951308

Expressions

- Arithmetic, logical, or string
- Enclose in **parentheses** when used as arguments to commands, **IF** statements, or **DO** conditions

```
volume create height $h radius1 $r radius3 $r \
              offset 0 0 ($h/2) zaxis frustum
```

Arithmetic Expressions

- Evaluate to numeric results
- FORTRAN-like syntax
  - *E1 op E2*
    where *E1* and *E2* can also be expressions, and *op* ***(refers to operators)*** is
    - **+** (addition)
    - **-** (subtraction)
    - ***** (multiplication)
    - **/** (division)
    - **^** (exponentiation, note difference from FORTRAN)
  - Order of operations is **^ * / + -**
  - Use parentheses to override

- Examples:
  - **$x + 10**
  - **-5.0 * $a / $b**
  - **3^3.5 + 4 * $y**
  - **(3^3.5 + 4) * $y**

Logical Expressions

- Evaluate to "true" or "false"
- FORTRAN syntax
  - E1 *.op.* E2
    where E1 and E2 are expressions, and
    .op. is
    - **.GT.** (greater than)
    - **.LT.** (less than)
    - **.GE.** (greater than or equal to)
    - **.LE.** (less than or equal to)
    - **.EQ.** (equal to)
    - **.NE.** (not equal to)
    - **.AND.** (true if both E1 and E2 are true)
    - **.OR.** (true if either or both are true)
  - **.NOT.** E1 (true if E1 is false)

- Examples:
  - `$x .lt. 5`
  - `$y .gt. 10`
  - `($a.eq.4).and.(($b+$c).lt.$d)`
  - `.not. $z`

String Expressions

- String parameters defined as
  `$name = "GAMBIT"`
- Enclose string constants in double-quotes
  - **`"volume.1"`**
  - **`"fluid"`**
- Concatenation: *str1* **`+`** *str2*
  - **`$base = "volume"`**
  - **`$extension = ".one"`**
  - **`$label = $base + $extension`** yields **`"volume.one"`**
  - **`$gam = "/usr/" + "gambit"`** yields **`"/usr/gambit"`**

Functions

- Function can be used in any expression
- Return a single numerical, logical, or string value
- Not case sensitive (with exceptions)
- Arguments are constants or expressions enclosed in parentheses
  - **ABS(***exp***)**
  - **COS(***exp***)**

String Functions

- Many string functions available, such as **STRLEN, STRCMP** and **CSTRCMP**
- **STRLEN:** number of characters in a string
  - **$x= STRLEN("title")⇒ $x=5**
- **STRCMP**: string compare (Case sensitive)
  - **$y= CSTRCMP ("ABD","abd") ⇒ $y=-1**
- **CSTRCMP**: case insensitive string compare
  - **$y= CSTRCMP ("ABD","abd") ⇒ $y=0**

Arithmetic Functions : Trigonometric

ACOS(*exp*)	arc-cosine
ASIN(*exp*)	arc-sine
ATAN(*exp*)	arc-tangent
COS(*exp*)	cosine
COSH(*exp*)	hyperbolic cosine
SIN(*exp*)	sine
SINH(*exp*)	hyperbolic sine
TAN(*exp*)	tangent
TANH(*exp*)	hyperbolic tangent

Arithmetic Functions : Miscellaneous

ABS(*exp***)**	absolute value
EXP(*exp***)**	exponential
INT(*exp***)**	integer truncation
LOG(*exp***)**	natural logarithm
LOG10(*exp***)**	base 10 logarithm
MAX(*exp1***,***exp2***)**	maximum of *exp1* and *exp2*
MIN(*exp1***,***exp2***)**	minimum of *exp1* and *exp2*
MOD(*exp1***,***exp2***)**	modulo (remainder) of *exp1*/*exp2*
POW(*exp1***,***exp2***)**	same as *exp1*^*exp2*
SIGN(*exp***)**	-1.0 if *exp* < 0, else 1.0
SQRT(*exp***)**	square root

Important String & Database Functions

NTOS(*exp*) Converts a **N**umber **TO** a **S**tring

Example: `If $i = 1:`

`"wall."+ NTOS ($i) = "wall.1"`

LASTID(*tag*) ID of last-created entity, *tag* =

`ve_id` or `1` (vertex)
`ed_id` or `2` (edge)
`fa_id` or `3` (face)
`vo_id` or `4` (volume)
`gr_id` or `5` (group)
`cs_id` or `6` (coordinate system)
`bl_id` or `7` (boundary layer)

Example: If five vertices has been created:

`LASTID(ve_id)` or `LASTID(1)` = 5

Useful Database Functions (1)

- **ARCLEN(edge)**
  - Returns the length of a specified edge
  - If no edge name is specified, ARCLEN returns length of the shortest edge

```
$X = ARCLEN("edge.17")
```

- **BBOX(entity)**
  - Returns array of six Cartesian coordinate values of diagonally opposed corners on a rectangular box that bounds individual entity (vertex, edge, face, volume, or group) or the entire model
  - The array values are reported in the order: xmin, ymin, zmin, xmax, ymax, zmax

```
$X = BBOX("volume.3")
```

  - If no entity name is specified, GAMBIT returns values of the box bounding the entire model

Useful Database Functions (2)

- **ENT2LOC(entity)**
  - Returns the coordinates of the center point of the ***entity***

```
$X = ENT2LOC("face.13")
```

- **LOC2ENT(return_type, x, y, z)**
  - Returns entity name in closest proximity to a specified coordinate location
  - 'return_type' specifies the type of entity to be located
  - (x,y,z) represent coordinates of the search point

```
$X = LOC2ENT(t_fa, 116, 57, 209)
```

- **RETLABEL(entity_type, n)**
  - Returns the last nth entity name used in the model for a specified entity type

```
$X = RETLABEL(t_ve, 2)
```

Useful Database Functions (3)

- **`LISTENTITY(`*`return_type, filter_type, filter_entity`*`)`**
  - Returns a string-array with the filtered list of entities, zone definitions, coordinate systems, boundary layers, or size functions of a specified type currently existing in the model

```
$X = LISTENTITY(t_ed, t_fa, "face.5")
```

    - returns array of all edges of face.5

```
$X = LISTENTITY("t_bl")
```

    - returns all boundary layer names

- For more functions and more details on any of these functions see:

file://~/fluent.inc/gambit2.0.4/help/html/users_guide/ug0b.htm

Replace the "~" with the installation path name on your system

Also change the appropriate **gambit-version** number

System Functions

- **FILEEXISTS(*filename*)**
  - Flag indicating the existence(1) or nonexistence(0) of a specified file

```
$X = FILEEXISTS("model_01.jou")
```

- **GETCWD()**
  - String for current working directory

```
$X = GETCWD()
```

- **GETENV(*env_variable*)**
  - Get value of the environment variable, ***env_variable***

```
$X = GETENV("GAMBITROOT")
```

- **UNAME()**
  - Name of current operating system

```
$X = UNAME()
```

Set Parameter by String Concatenation

```
/journal file for creation of a pipe of varying height
/parameter definition
/$h is the height of the pipe
$h= 6.4
     •••
/commands for the creation of the pipe, meshing and
/definition of boundary zones
     •••
/commands to export the mesh
solver select "FLUENT 5/6"
$title = "pipe-"
$end = ".msh"
$id = $title + NTOS ($h) + $end
export fluent5 $id
```

/This journal file will export a file named: **pipe-6.4.msh**

FIDAP users:

```
solver select "FIDAP"
$end = ".FDNEUT"
export fidap $id
```

Exported file: pipe-6.4.FDNEUT

DO Loops (1)

- Syntax
  - **DO PARA "$***param***" INIT** *exp1* **COND(***cond***) INCR** *exp2*
    commands
    ENDDO
- Where
  - **PARA -** loop parameter
    - **$***param* - must be defined before loop
    - Its value is overwritten by the initialization of the DO Loop
  - **INIT -** initial value of the loop parameter
  - **COND -** condition
    - Example: **(***cond***)** = **($param .le. 5)**
  - **INCR -** increment
  - **INIT** and **INCR** are optional; if one of them is not defined, its value is set to 1 (i.e. **$***param* is initialized to be 1 or is incremented by 1)

DO Loops (2): Example

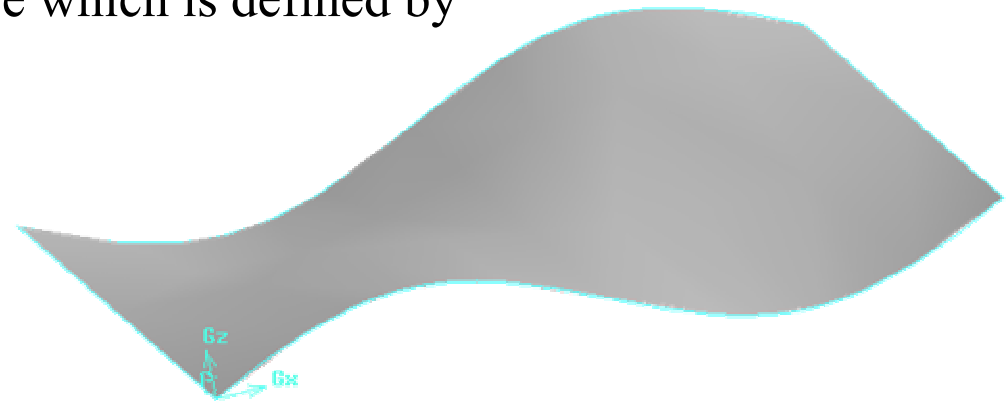
- The following GAMBIT journal creates 36 vertices at every integer position in the x-y plane, where $0 \leq x,y \leq 5$

```
$i = 0
$j = 0
$imax = 5
$jmax = 5
do para "$i" init 0 cond ($i .le. $imax)
do para "$j" init 0 cond ($j .le. $jmax)
    vertex create coordinates $i $j 0
  enddo
enddo
```

DO Loops (3): Example

- The following GAMBIT journal creates a set of grid points (9 x 9) which are used to approximate a surface which is defined by

$$z = .15\sin(\pi x)\cos(\pi y/2)$$




```
$i    = 0
$imax = 2
$j    = 0
$jmax = 2
$inc  = .25
$fact = .15
do para "$i" init 0 cond ($i.le.$imax) incr $inc
do para "$j" init 0 cond ($j.le.$jmax) incr $inc
  vertex create coordinate $i $j ($fact*sin(RAD2DEG*PI*$i)\
                                 *cos(RAD2DEG*PI*$j/2))
 enddo
enddo
$vertices = LISTENTITY(t_ve)
face create vertices $vertices rowdimension 9
```

IF-THEN-ELSE Blocks (1)

- Syntax
  - **`IF COND (`***exp*`)`**
    true-commands
    `ELSE`
    false-commands
    `ENDIF`
- Where
  - **`COND -`** condition
    - Example: **`(`***exp*`)`** = **`($param .le. 5)`**
  - **`ELSE`** and *false-commands* are optional
  - Can be nested
  - No **`ELSEIF`** defined (must use nested **`IF`**)

IF-THEN-ELSE Blocks (2)

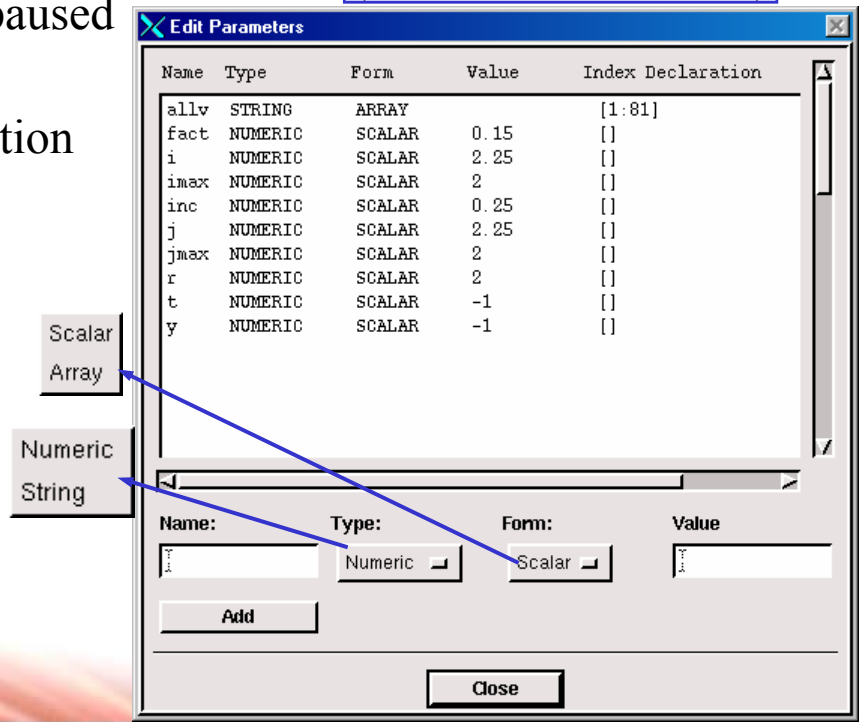
- In the following Gambit journal the condition is false and a coarse grid is created

```
/coarse grid: a = - 1
/fine grid:   a = 1
$a= -1
if cond ($a .gt. 0)
 volume mesh "volume.1" cooper source "face.1" \
                                      "face.3" size 1
else
 volume mesh "volume.1" cooper source "face.1" \
                                      "face.3" size 10
endif
```

Current Limitations

- Parameter definition in the **Edit - Parameters** form does not produce journal commands
- Parameters and expressions can **NOT** be used within the GUI
- Journals produced by GAMBIT contain the ***values*** of parameters and expressions, **not** the parameters/expressions themselves
- Batch execution of Journal files can be paused but not cancelled or revoked
- **`Control-C`** (^C) or any other interruption is not available

Edit – Parameters Form
(Edit – Parameter menu item)



Steps to Parameterize GAMBIT Journals

- Build initial model with GUI
  - First use a set of basic numerical values
  - Mesh model and specify Boundary Types
  - Save journal file with unique name
- Editing the journal file:
  - Define key parameters at the top of the file and include comments
  - Replace values with parameters throughout
- Check the journal file:
  - Replay the journal to make sure that parameters were defined and used correctly
  - List of all parameters and their current values can be checked (Edit-parameters form)

Additional Info on Journals

- Journal from any dbs can be restored by: **`gambit a -res b.jou`**
- A journal file can call (nested) another journal file

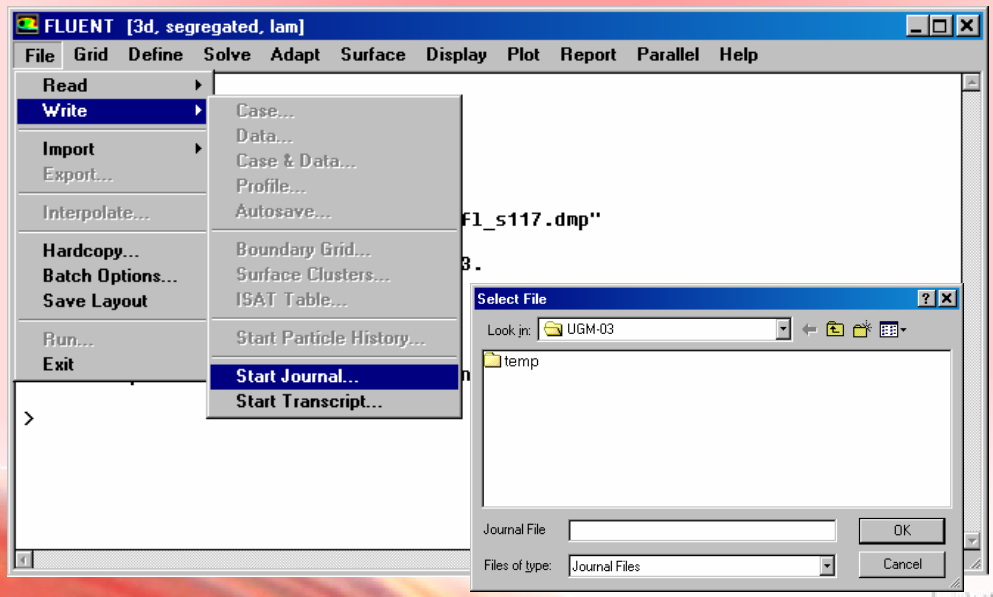
  `READ FILE "small.jou"`
- A journal file can be written in one of formats: original name, last-id based or location based
  - The corresponding variable, **`JOURNAL_ENTITY`**, can be set on the "Edit Defaults" form as:

    0 = org. name, 1 = LASTID, 2 = Location based
- Make sure to identify and limit ranges to retain topological integrity to the original journal
  - If the topology changes for a "valid" range of parameter values, separate journals need to be maintained corresponding to each valid topology

Summary of Journal File Uses

- Parameterized journals can save large amounts of time for parametric studies
- The **DO** loops and **IF-ELSE** blocks can be used to control events in the journal file
- Time spent up-front thinking about how to best parameterize your journals can save time later in the process
- GAMBIT journal files can be combined with FIDAP journal files.
  - allows parameters to be defined only once if any of the boundary conditions depend on the parameterized geometry.

Fluent Journals



Journals in Fluent

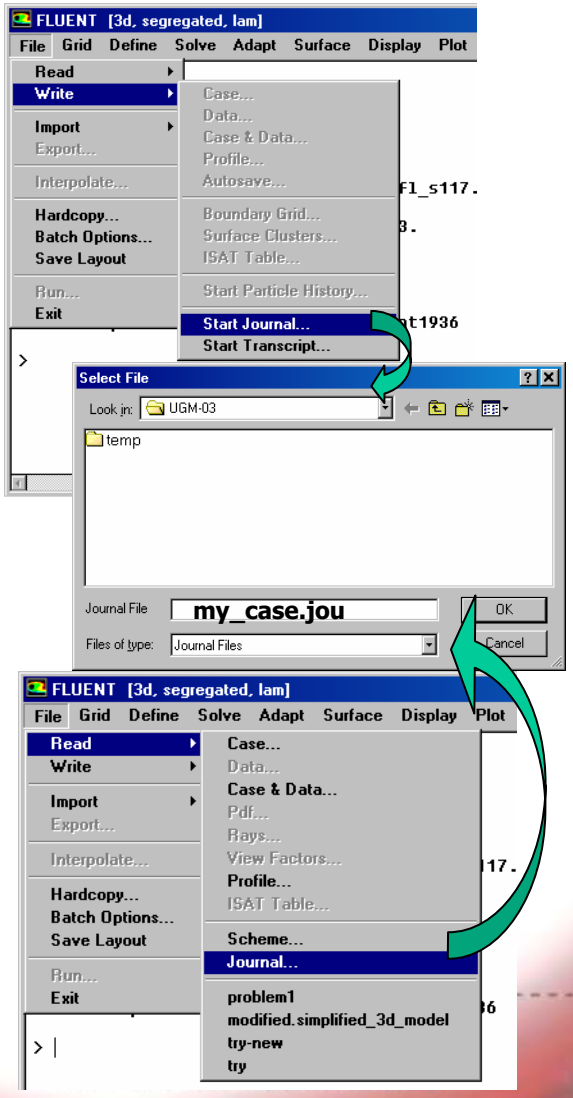
- A fluent journal can be created in 2 ways
  - A cortex based journal recording from the **GUI**
  - Writing clean short easy-to-follow journal using **TUI**
- We will briefly show how to record GUI journal / macro
- TUI based journal creation is more reusable and editable
- A brief introduction to Fluent Architecture would be useful
- Let's go with the flow of setting up a case, running and post-processing

Fluent Architecture

- Note that Fluent is build on a custom client-server architecture
  - It has a user-interfacing process, **cortex**, (I/O, post-processing, solver-setting)
  - And a **solver** process (can be several processes for parallel execution)
  - Both GUI and TUI interfaces are available to communicate between the cortex and the solver
  - Like gambit creation of journal and transcript files are possible
  - Unlike gambit, journals and transcripts are not created by default – user has to choose to create the journal and/or transcripts
- In Fluent GUI lingo, "journal" is interchangeable with "macro"
  - Both follows the syntax of SCHEME – an interpreter based language
- Description of 'Scheme' is kept beyond the scope of this presentation

Recording & Reusing the Journal

- To record the entire case setup, running the case, and post-processing, follow the sequence:
  - Start appropriate Fluent version
  - Visit **`File-Write-Start_Journal`** menu and provide a journal file name (say, my_case.jou)
  - Go about reading the mesh, setting the case, iterating, case and data saving and even post-processing
  - Visit **`File-Write-Stop_Journal`** menu (this will save my_case.jou on the disk)
- Remember to move the case & data file to other names, before reusing the my_case.jou for a new mesh
  - The new mesh file should have the same name as the previous one
  - Visit **`File-Read-Journal`** menu and read in my_case.jou
  - This will follow the foot step of the previous run



The Cortex Journal File

- Although, the syntax is quite repetitive, it is not very lucid
- Also, useful editing the file is not very easy
  - Change the mesh file name
  - Change fluid properties and/or boundary conditions
  - Change case, data file names
  - Add extra post-processing

The my_case.jou File

```
(cx-gui-do cx-activate-item "MenuBar*ReadSubMenu*Case...")
(cx-gui-do cx-set-text-entry "Select File*Text" "problem1.msh")
(cx-gui-do cx-activate-item "Select File*OK")
(cx-gui-do cx-activate-item "MenuBar*GridMenu*Scale...")
(cx-gui-do cx-set-list-selections "Scale Grid*Frame3(Units Conve
(cx-gui-do cx-activate-item "Scale Grid*Frame3(Units Conversion)
(cx-gui-do cx-activate-item "Scale Grid*PanelButtons*PushButton1
(cx-gui-do cx-activate-item "Scale Grid*Frame3(Units Conversion)
(cx-gui-do cx-activate-item "Scale Grid*PanelButtons*PushButton2
(cx-gui-do cx-activate-item "MenuBar*DefineMenu*Materials...")
(cx-gui-do cx-set-text-entry "Materials*Table1*Frame1*Table1*Tex
(cx-gui-do cx-set-real-entry-list "Materials*Frame2(Properties)*
(cx-gui-do cx-set-real-entry-list "Materials*Frame2(Properties)*
(cx-gui-do cx-activate-item "Materials*PanelButtons*PushButton1
(cx-gui-do cx-activate-item "Question*OK")
(cx-gui-do cx-activate-item "Materials*PanelButtons*PushButton1
(cx-gui-do cx-activate-item "MenuBar*WriteSubMenu*Case...")
(cx-gui-do cx-set-text-entry "Select File*Text" "problem1.cas.gz
(cx-gui-do cx-activate-item "Select File*OK")
(cx-gui-do cx-activate-item "MenuBar*DefineMenu*Boundary Conditi
          ⋮
(cx-gui-do cx-set-text-entry "Select File*Text" "problem1.cas.gz
(cx-gui-do cx-activate-item "Select File*OK")
(cx-gui-do cx-activate-item "Warning*OK")
(cx-gui-do cx-activate-item "MenuBar*WriteSubMenu*Stop Journal")
```

The TUI Journal File

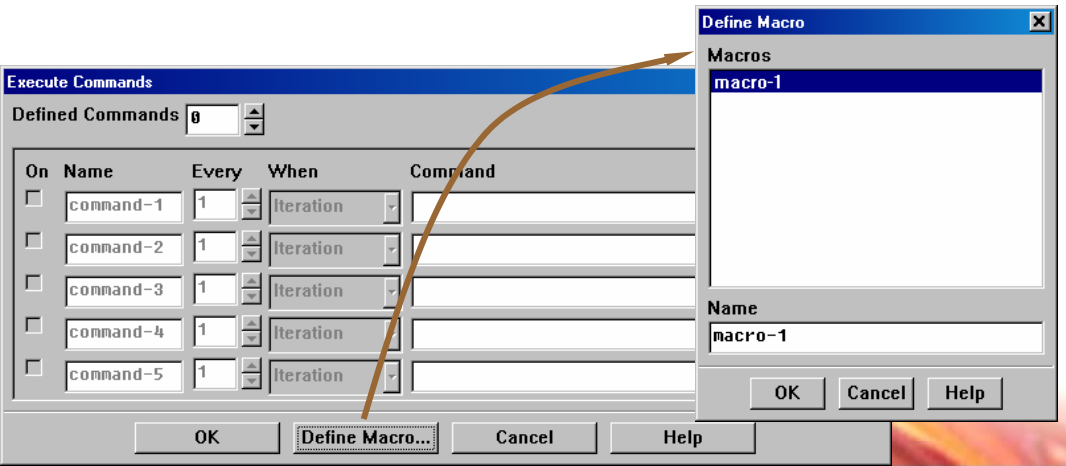
- These files can be generated even before actually setting up the first case itself
- TUI based journal for the previous case can be rewritten as follows:

```
file read-case problem1.msh
grid scale 0.0254 0.0254 0.0254
define mat cc air poly y , 1000 n n y , 0.1 n n n n n y
define bc mfi mf-in y 0.05 n 0 n y y y
define bc po pr-out n 0 n n y
sol ini ini
it 1000
file wcd problem1.cas.gz y
exit y
```

- The above is an exact replacement of the cortex based journal shown earlier
- However, the user needs to study and write the TUI based commands in this case
  - Get the complete list of TUI commands at: **http://www.fluentusers.com/fluent6/doc/ori/html/tuilist/main_pre.htm**
- To replace various names, boundary conditions and material properties is very easy

Part Journals / Macros

- Fluent **`TUI`** journals can be written for part of the job
- Similarly, **`GUI`** based journals can be written out for part of the job too
- Note that these journals are written out as a separate file (than case file)
- A macro on the other hand can be written from the **`Solve-Execute_command`** menu
- Unless specifically saved using TUI command (**`file write-macro`** ***`filename`***), these macro-s are available only in the current fluent session
- To read back a previously saved macro, use TUI command: **`file read-macro`** ***`filename`***



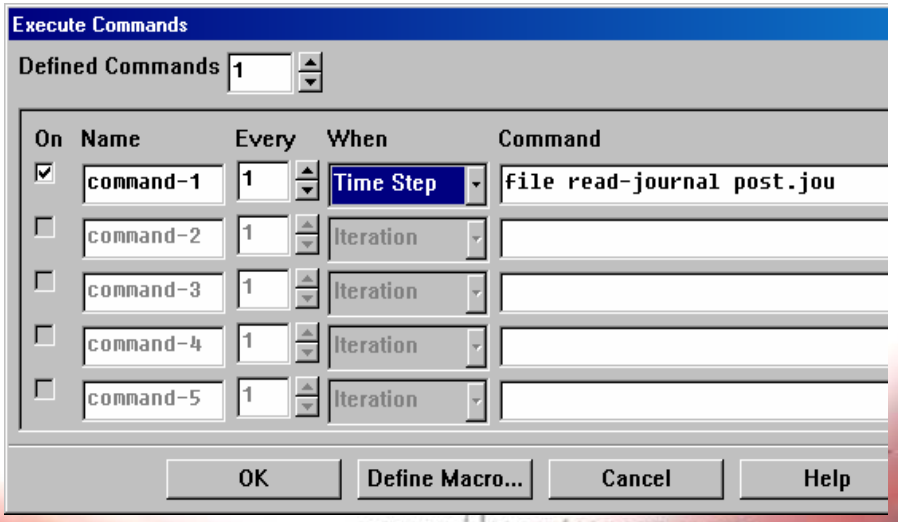
- These macro-s have exactly identical syntax and functions as the GUI journals

Some TUI-Journal Examples

- Post processing with such journals
- The commands can be figured out in one of the two ways:
  - Visit the TUI command list on the help/documentation
  - Try out the commands from the root_command, e.g., **`Display`**
- Aspects of post-processing are covered in a separate presentation
- Such TUI journals are useful even for interactive runs for flawless post-processing
  - Visit **`Solve-Execute_command`** menu and set up a command to be executed at chosen frequency of iteration or timestep and write the command as:

    `file read-journal post.jou`

```
dis set hard-copy x-r 0
dis set hard-copy y-r 0
dis set hard-copy  landscape no
dis set hard-copy  color-mode color
dis set g-s 3 4 6 1 5 ()
dis set g-z ()
dis set f-g y
dis set r-g y
dis part vel "injection-0" () , ,
dis view restore-view view-0
dis set colors background "white"
dis hard-copy fil-dpm-%t.tif
dis set colors background "black"
```



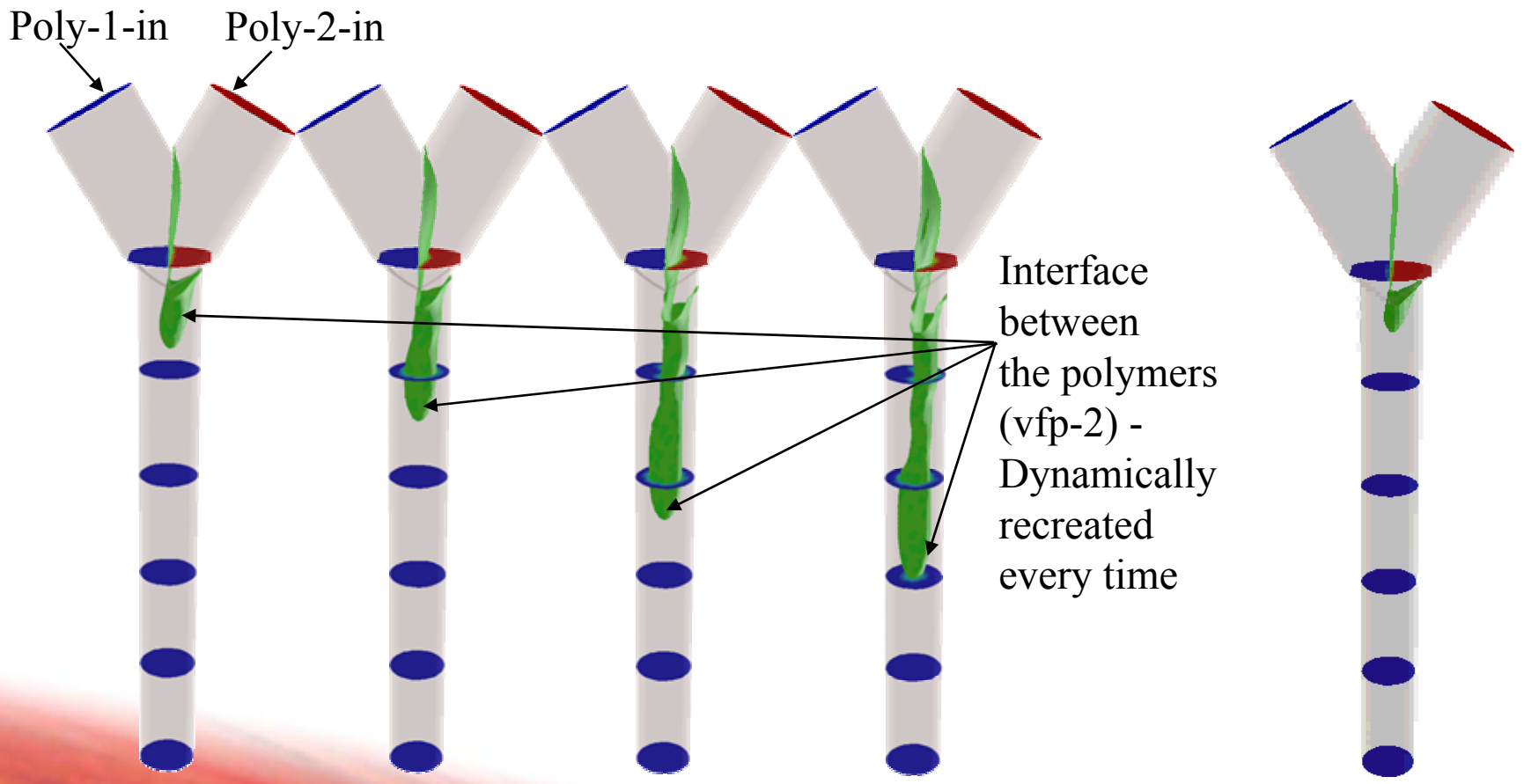
Details of a Journal for Graphics

- A sample journal file (post.jou) would contain:

```
display set hc color color          ; Select 'color' for hardcopy
dis set hardcopy driver tiff        ; Select 'tiff' format
dis set hc invert-background? y     ; Check 'on' white-back-ground
dis set hc landscape? n             ; Check 'off' landscape
dis se hc x-r 800                   ; Select x-resolution
dis se hc y-r 1200                  ; Select y-resolution
surf iso-surf vof-poly-2 vfp-2 , .5 ,; Create iso-surface to locate
                                    ; the interface (vfp-2)
dis set win axe vis no              ; Make Axis invisible
dis set win sca vis no              ; Make legend invisible
dis set con surf vfp-2 outlet       ; Select surfaces to plot
     poly-1-in poly-2-in y=0.05     ; contours on
      y=0.1 y=0.15 y=0.2 y=0 ,
dis vi res view-0                   ; Restore the view, view-0
dis con vof-poly-2 ,,               ; Display contours of VOF of Poly-2
dis hc vof-%t.tif                   ; Create graphic files on disk
dis set hc invert-background? y     ; Restore background as black
surf ds vfp-2                       ; Delete the VOF iso-surface
```

View-0 must be predefined

Some frames ...


Poly-1-in
Poly-2-in
Interface between the polymers (vfp-2) - Dynamically recreated every time

Some Enhancements on the Journal

- Create a filename within the journal

```
(define aaa "animate")
```

- Check and if not exists, create a directory to hold graphic files
  - *Works for both UNIX & WINDOWS*

```
(define d_name "animate")
(if (nt?)
(system (format #f "\"if not exist ~a mkdir ~a\"" d_name d_name))
(system (format #f
"\#\!/bin/sh\n if test -d ~a \; then echo \"\"\; \n else mkdir ~a
\n fi"  d_name d_name)))
```

- Create hardcopy of the graphic in the d_name directory
  - *Works for both UNIX & WINDOWS*

```
(if (nt?)
(ti-menu-load-string (format #f "dis hc  ~a\\~a.tif " d_name aaa))
(ti-menu-load-string (format #f "dis hc  ~a\/~a.tif " d_name aaa)))
```

Another Example

- Often we run transient cases with auto-saving of data at some time-intervals and intend to do post-processing at a later time
- This could be very demanding, if not tedious
- A journal can significantly reduce the task and enhance reusability
  - You may reuse the journal for another case
  - May generate a different set of post-processing
  - Can systematically run Define_on_Demand routines with each data set to perform numerical analysis at each saved time
  - Generate transient plot of certain parameter(s)
- We will cursorily introduce Scheme here to facilitate automatic reading of the data file sets and dumping appropriately named graphic as well as xy-data files

Post-Analysis of Transient Data

- Note that the "auto-saved" data file names are created as a concatenated base-name and the time-step number
  - e.g., my_case0000.dat.gz, my_case0010.dat.gz etc. when the autosave data frequency was 10
- We will construct a "do-loop" in scheme to first construct such filenames and then read those in and eventually perform the post-processing activities with TUI as illustrated earlier
- The process is as follows:
  - Start appropriate Fluent version
  - Read in the Case file
  - Read in the journal file to read successive data files, do post-processing and save graphics and other files

Journal for Transient Data

```
(define (my-post-proc);;name of the journal-function
(define basename "mycase") ;;Filename prefix for the data files
(define nstart 598000);; time-step # for the 1st data set
(define nsave 2000)    ;; auto-save frequency
(define ndata 4)       ;; # of data-sets
(do ((i nstart (+ nsave i)))((= i (+ nstart (* ndata nsave))))
  (begin
  (ti-menu-load-string (format #f "f rd ~a~d.dat.gz" basename i))
  (ti-menu-load-string (format #f "dis set grid-zones  3 4 5 6 8 9  ()"))
  (ti-menu-load-string (format #f "dis set filled-grid? yes"))
  (ti-menu-load-string (format #f "dis set contour surface ()"))
  (ti-menu-load-string (format #f "dis con wax vof 0 1"))
  (ti-menu-load-string (format #f "dis view restore-view view-1"))
  (ti-menu-load-string (format #f "dis set hard-copy x-r 0"))
  (ti-menu-load-string (format #f "dis set hard-copy y-r 0"))
  (ti-menu-load-string (format #f "dis set hard-copy  landscape no"))
  (ti-menu-load-string (format #f "dis set hard-copy  color-mode color"))
  (ti-menu-load-string (format #f "dis set colors background \"white\" "))
  (ti-menu-load-string (format #f "dis hard-copy ~a-~d.tif" basename i))
  (ti-menu-load-string (format #f "dis set colors background \"black\" "))
  )))
```

Some Details on The Journal

- Journal can be loaded in a Fluent session through **File-Read_Journal** menu

```
(define (my-post-proc)…
```

- The function can be executed by typing `(my-post-proc)` in the Fluent console window
- The do-loop syntax illustrates the construct for processing multiple dataset

```
(do    ((i nstart (+ nsave i)))
       ((= i (+ nstart (* ndata nsave))))
 (begin.....))
```

- The following function is the most widely used wrapper for a TUI command

```
(ti-menu-load-string (format #f "f rd ~a~d.dat.gz" basename i))
```

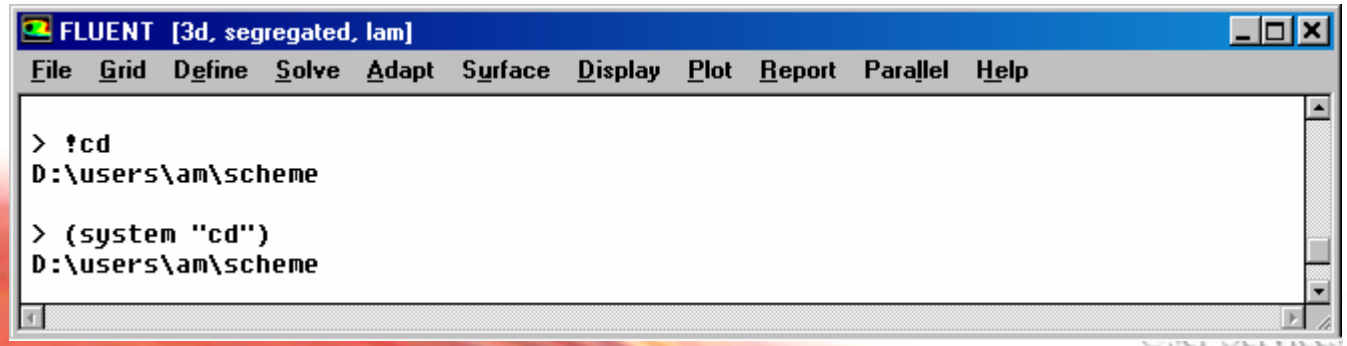
- The format function allows reconstructed command based on user-specified values
  - In this case a TUI command, e.g., **f rd mycase0010.dat.gz**, is being created
  - Note that both basename and i are used as variable to construct the TUI

TUI Resource

- For a complete list of TUI commands, look in the help area on your system:

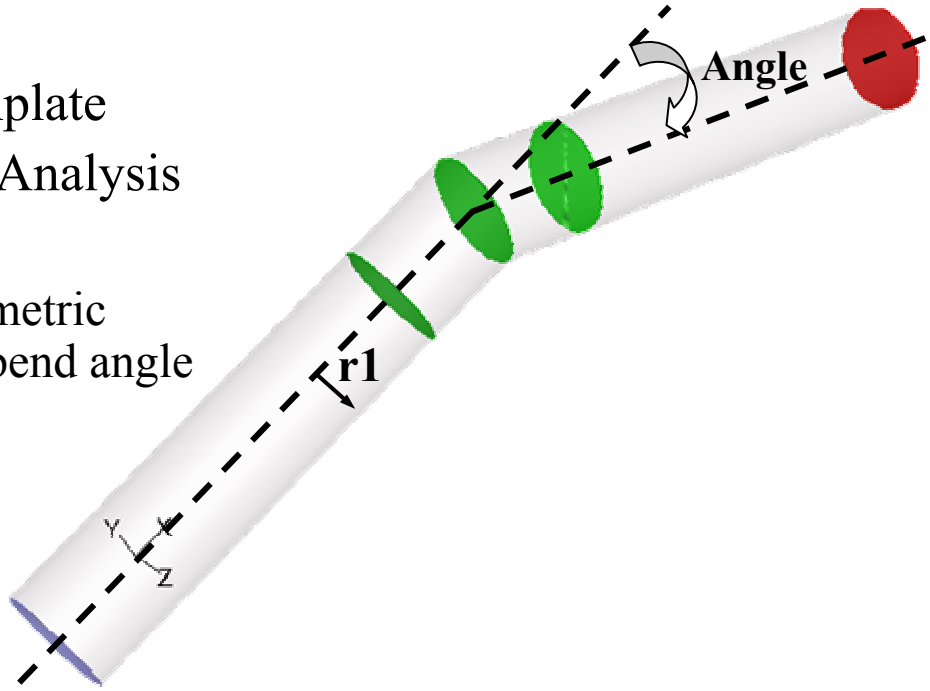
```
file://~fluent.inc/fluent6.1/help/html/tuilist/main_pre.htm
```

  - Where the "~" is the full path name for your installation of the Fluent.Inc directory
- Note that a system command can be executed from the Fluent console window by preceding the command with a '!' or as (system command) where command is a valid system command in quotes ("…")



Drive Gambit/Fluent Thru' Excel

- Use Excel to create your own template
- Here is one example: Bend Flow Analysis
- This is a cook-up problem
  - Say the user wants to study parametric effects of the pipe diameter and bend angle vis-à-vis material properties
- The task will include:
  - Creation of Clean and parameterized Gambit journal
  - Creation of Fluent Journal to read in the mesh and run the case as well as write out relevant files
  - Customize Excel to take user-inputs and drives the journals in the background



Drive Gambit/Fluent Thru' Excel

- Note the gambit journal begins with the redefinition of the default parameter GUI.GENERAL.TRANSCRIPT so that background run does not throw the transcript on the dos window
- Also note that the excel inputs are redirected to a SSV file named, **"geom-input.prn"** and read in the journal file
- Rest of the journal file is like any other typical gambit journal file

```
default set "GUI.GENERAL.TRANSCRIPT" numeric -1
/===== User inputs ==========================
read file "geom-input.prn"
/======================End of User inpus======
$height=10*$r1
$is=$r1/50
$r2=2*$r1
$offset=0.5*$height
$hb2=$height/2
$ab2=$angle/2
$r8=$r1*8.0
$in1=30
$in2=20
$in3=40
$in4=24
$b1=$r1/100
$bgf=1.2
$br=14
$h4=$height*4
$d1=$r1 * (1-cos($angle))
$d2=$r1 * sin($angle)
$d3=2 * $r1 * (1-cos($angle))
/-----------------------------------------------
volume create height $height radius1 $r1 radius3 \
                $r1 offset $hb2 0 0 xaxis frustum
volume move "volume.1" offset -$height 0 0
volume copy "volume.1" to "volume.2"
```

• • •

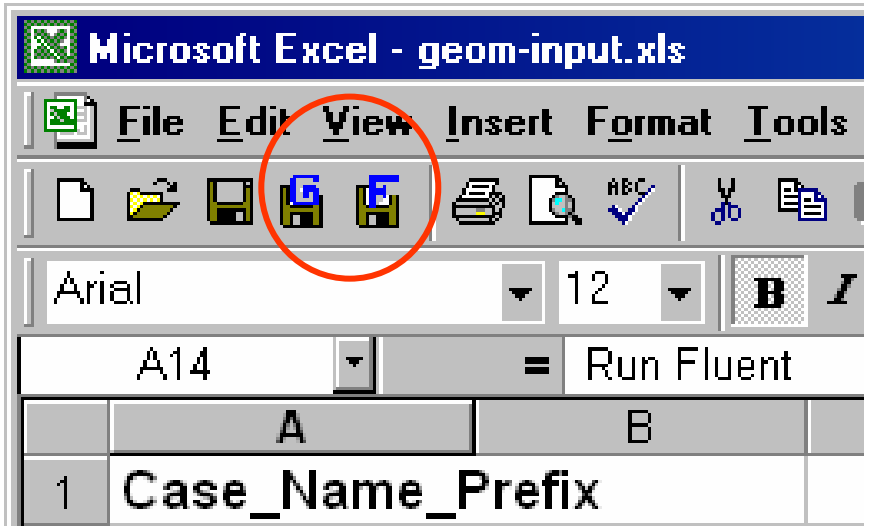
Drive Gambit/Fluent Thru' Excel

- Fluent inputs from excel are stored in the "flu-input.prn"file
- The function to open the file for input is open-input-file
- (%read ifile) function reads in one field at a time and the value can be redirected to appropriate variable
- In rest of the fluent journal, regular use of `ti-menu-load-string` and `format` statements will be made to custom use of TUI

```
(define ifile (open-input-file "flu-input.prn"))
(%read ifile)(%read ifile)
(define st-file  (%read ifile))      ;; Prefix of mesh filename
(%read ifile)
(define sx (%read ifile))            ;; scale factor for length
(%read ifile)
(define density (%read ifile))       ;; density
(%read ifile)
(define sp-heat (%read ifile))       ;; specific-heat
(%read ifile)
(define th-cond (%read ifile))       ;; thermal conductivity
(%read ifile)
(define vis (%read ifile))           ;; viscosity
(%read ifile)
(define m-in (%read ifile))          ;; Inlet mass flow rate
(%read ifile)
(define t-in (%read ifile))          ;; Inlet (total) T at P-in
(%read ifile)
(define p-out (%read ifile))         ;; Static P at P-out
(%read ifile)
(define t-out (%read ifile))         ;; Backflow T at P-out
(%read ifile)
(define t-wall (%read ifile))        ;; Wall T
(%read ifile)
(define num-iter (%read ifile))      ;; No. of iteration
```

Customizing EXCEL

- Couple of macros need to be recorded to store the SSV files for gambit and fluent inputs
- Appropriate icons can be created for subsequent easy use
- The macros can be stored with PERSONAL.XLS file so that they are available on any subsequent excel session



Customizing EXCEL

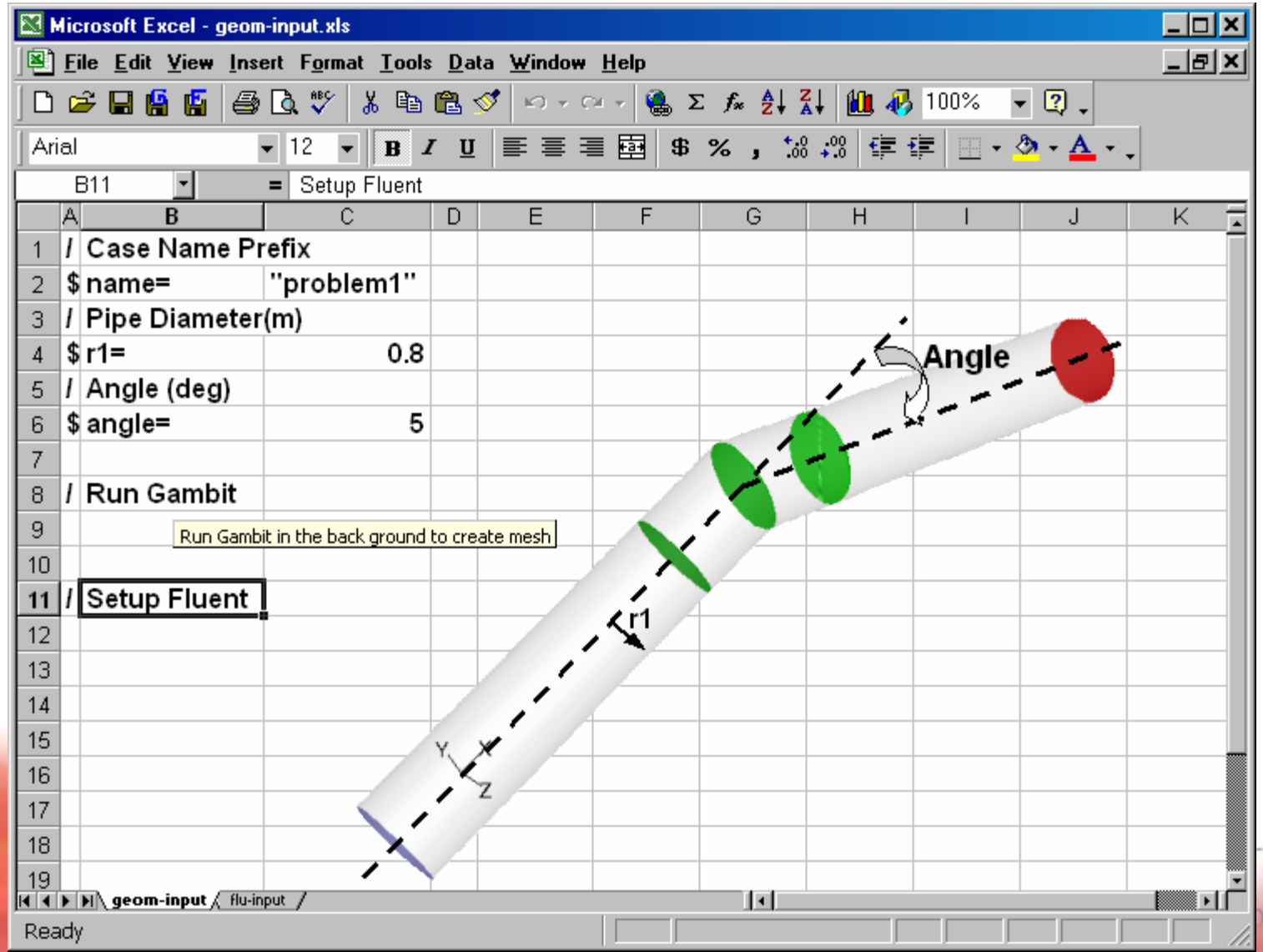
- Small batch files for Fluent and Gambit will be required to make clean start and redirect outputs
- Gambit Batch called from Excel

```
del default-id*
del *lok
gambit -inp mb-excel.jou
```

- Fluent Batch called from Excel

```
del output
fluent 3d -g -i f-excel.jou -o output
```

Gambit Template on Excel



Fluent Template on Excel

Microsoft Excel - geom-input.xls

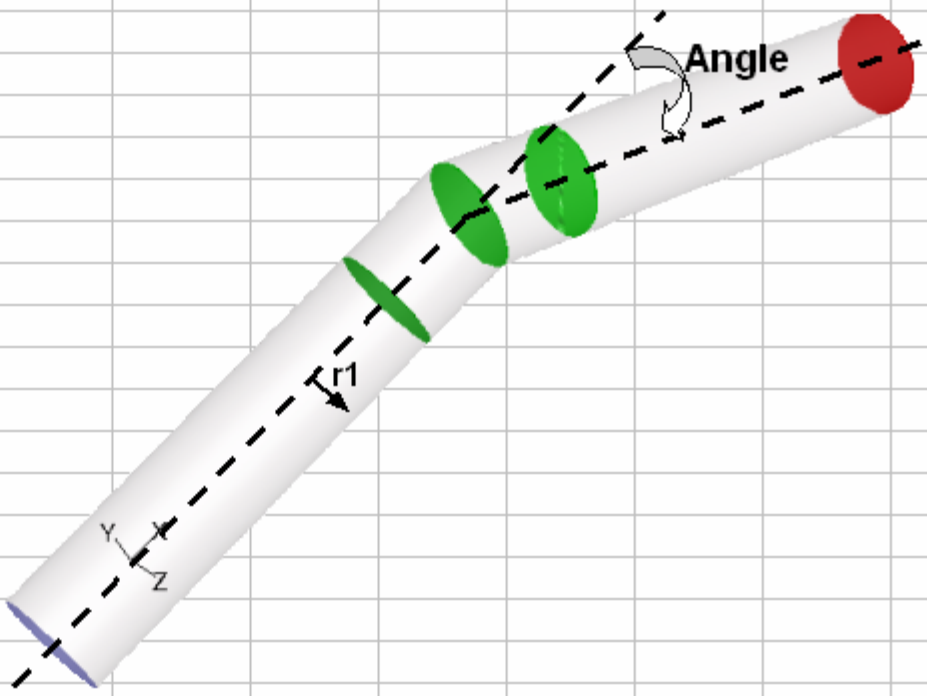
File Edit View Insert Format Tools Data Window Help

Arial 12 100%

A14 = Run Fluent

	A	B
1	Case_Name_Prefix	
2	Name	"problem1"
3	Scale	0.0254
4	Density	1000
5	Specific_Heat	1000
6	Thermal_Cond.	20
7	Viscosity	0.1
8	Mass_Flow-in	0.05
9	Temperature-In	300
10	Outlet_Pressure	0
11	Outlet_Temp.	500
12	Wall_Temp.	550
13	No_of_Iteration	300
14	Run Fluent	
15		
16		
17	Go Back to Geometry	
18		
19		

Run Fluent in the back ground

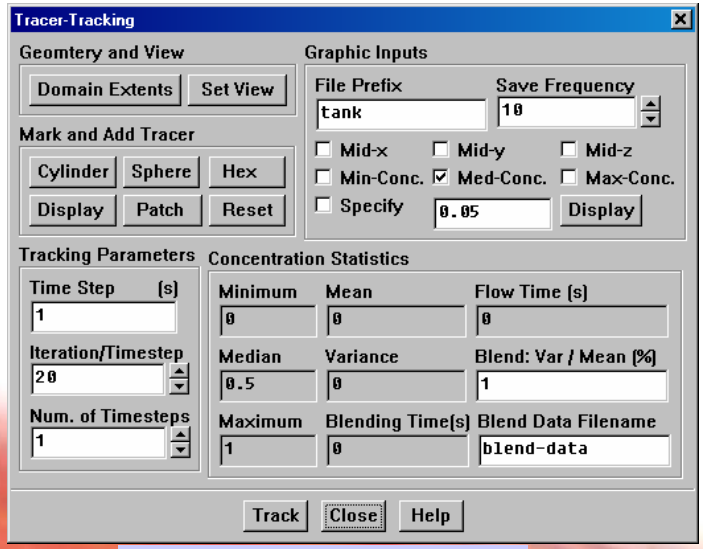


geom-input flu-input

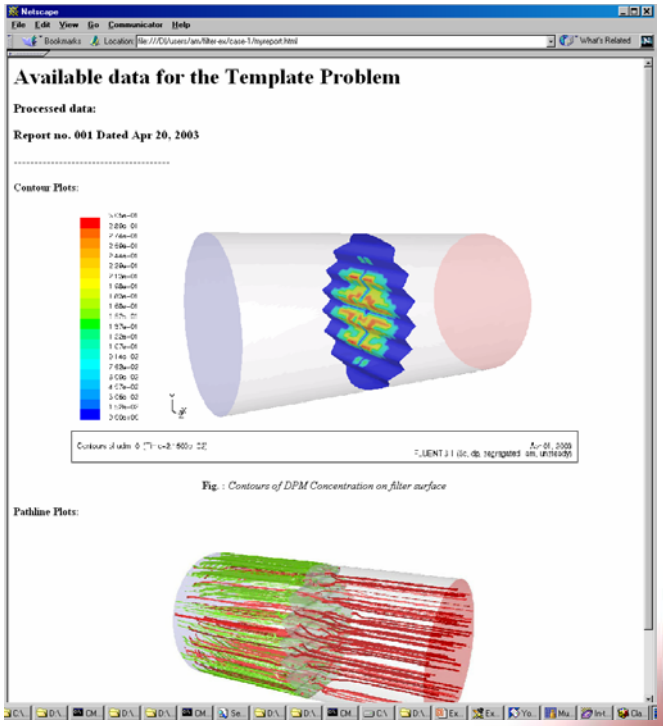
Ready

Templates With and Beyond Excel

- Template capabilities using Excel can be significantly enhanced with meaningful programs with Excel
- However, in all likelihood, this will provide passive access to Fluent/Gambit
- For more interactive templates, Fluent provides consulting services for custom front-ends



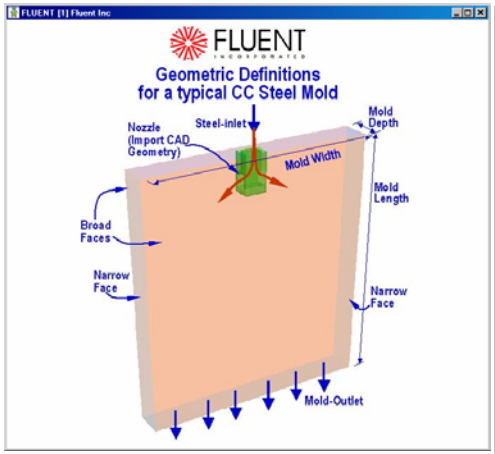
Blending Analysis

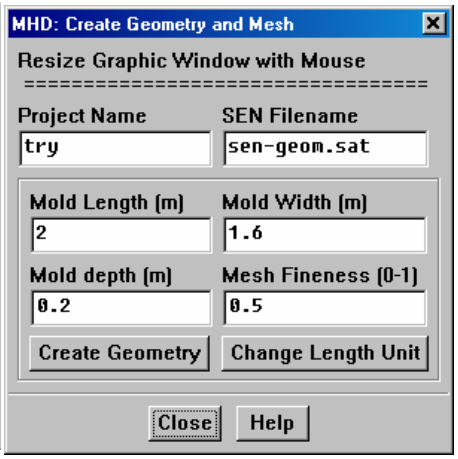


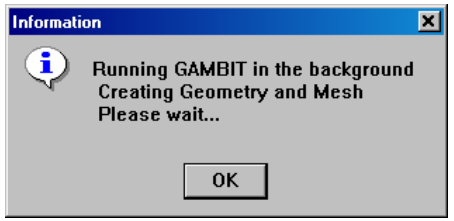
HTML Reports

Fluent Based Complete Template

- Fluent based fully functional templates are also possible for high volume repeat tasks with too many complex steps in problem setup and analysis







Closure

- The power of journals with both Fluent and gambit is getting richer with every release
- Increasing use of CFD as a simulation and model analysis tool drives the need for template creation
- This lecture intended to provided enough information to getting started with journal creation
- Information is also provided on how to get more detailed information on topics that could not be covered herein
- Additional information is appended
- Some neat examples of increasing complexity have been demonstrated
- Work with your support engineer in case you need further inputs or want us to develop templates for you

Appendix

- In the following few slides, additional information about custom GUI creation, menu addition and some special variable declarations are discussed
- In the interest of time during the short presentation, these topics may not be covered
- However, the slides are self-explanatory and for any further assistance please contact us

Some Details on The Journal

- You can add arithmetic expressions:

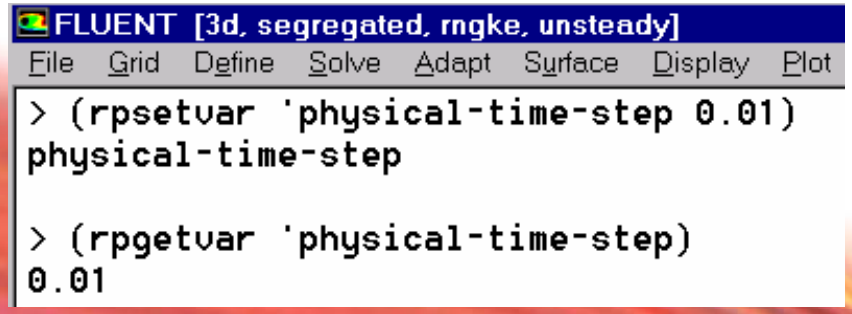
```
(define a (+ 20 3))
(define b (* a (/ 20.2 30)))
```

- Global variables in FLUENT are called '**RP_VAR**'s
  - These variables are present in all TUI, GUI and solver/UDF environments
  - **RP_VAR-s** may not be available for post-processing

! A '-' is allowed with variable / function names in scheme functions

- Local variables in scheme functions are not accessible from UDFs
- Local variables in UDFs are not accessible from scheme functions
- The RP_VARs are available from the scheme as:

(rpgetvar 'physical-time-step) returns Δt from FLUENT
(rpsetvar 'physical-time-step 0.01) sets Δt in FLUENT



RP_VARs

- The RP_VARs are available from UDF-s using a C-macro call
  - Example of **C**-calls:

```
f_time = RP_Get_Real("physical-time-step");
n_time = RP_Get_Integer("time-step");
```

- Available macro-s in C for defining RP_VARs:
  - `RP_Set_Real("var1",value);`
    `/* value (real) contains value of 'var1' */`
  - `RP_Set_Integer("var2",ivalue);`
    `/* ivalue (integer) contains value of 'var2' */`
- Use '`%rp-var-value`' function in scheme to access **var1** & **var2**:
  - `(define new-var1 (%rp-var-value 'var1))`
  - `(define new-var2 (%rp-var-value 'var2))`

! Note: Integer & real are accessed by same scheme function '`%rp-var-value`'
These macros require use of 'compiled' UDFs only

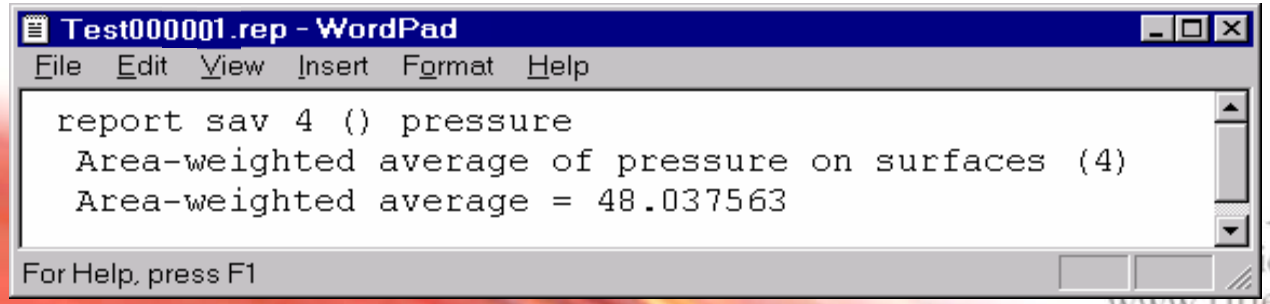
Reports to File

- In FLUENT, the report menu items have no 'write-to-file' option
  - Following journal function lets you do so:

```
(define test (format #f "test~06d.rep" (rpgetvar 'time-step)))
    (with-output-to-file test (lambda ()
      (ti-menu-load-string "report sav 4 () pressure")))
```

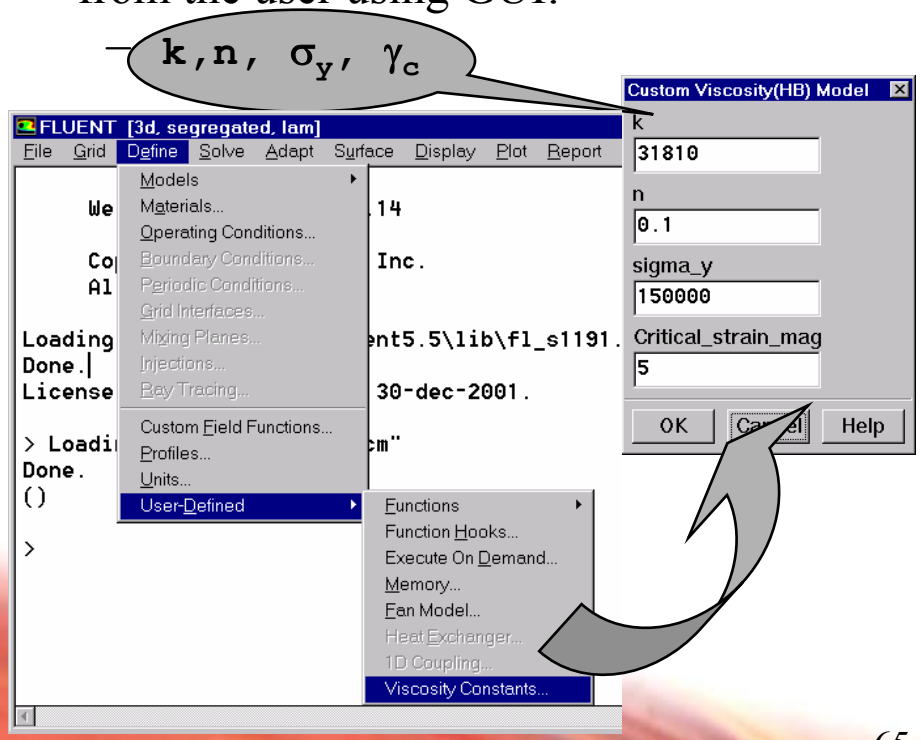
- Note:
  - with the '**format**' function, the file name is constructed (e.g., **test00001.rep**)
  - '**with-output-to-file**' function redirects FLUENT output to file
  - '**ti-menu-load-string**' function transmits TUI commands through the scheme
  - You need to load this function in FLUENT and can execute through solve-monitor-command at any chosen interval of time-steps by inserting the load command for the scheme function:

file read-macro write-report-to-file.scm



Custom Menu/GUI Using Scheme Functions

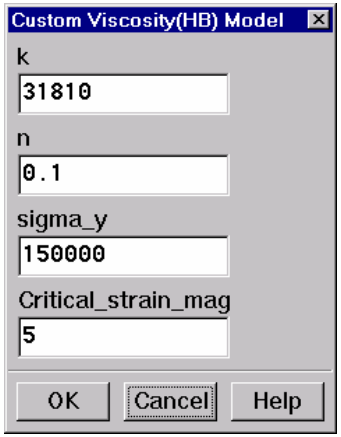
- Let us implement Herschel-Bulkley Viscosity model in fluent
- We will use **DEFINE_PROPERTY** routine to calculate the viscosity with 4 inputs from the user using GUI:




```
/*  UDF for Herschel-Bulkley viscosity */
#include "udf.h"
real T,vis, s_mag, s_mag_c, sigma_y,n,k;
real C_1 = 1.0;
real C_2 = 1.0;
real C_3 = 1.0;
real C_4 = 1.0;
int ia ;
DEFINE_PROPERTY(hb_viscosity,c,t)
{
T=C_T(c, t);
s_mag = CELL_STRAIN_RATE_MAG(c,t);
/* Input parameters for H-B Viscosity */
if (ia==0.0)
{ C_1 = RP_Get_Real("c_1");
  C_2 = RP_Get_Real("c_2");
  C_3 = RP_Get_Real("c_3");
  C_4 = RP_Get_Real("c_4");
  ia = 1;}
k       = C_1 ;
n       = C_2 ;
sigma_y = C_3 ;
s_mag_c = C_4 ;
if (s_mag < s_mag_c)
{vis = sigma_y*(2-
    s_mag/s_mag_c)/s_mag_c+k*((2-n)+(n-
    1)*s_mag/s_mag_c)*pow(s_mag_c,(n-1));}
else
{ vis = sigma_y / s_mag + k*pow(s_mag, (n-
    1));}
return vis;
}
```

Scheme Function for HB Viscosity Inputs

```
;;;; Create rpvars for user defined custom viscosity  model if they don't exist.;;
(if (not (rp-var-object 'c_1))(rp-var-define 'c_1 31810.0 'real #f))
(if (not (rp-var-object 'c_2))(rp-var-define 'c_2 0.1 'real #f))
(if (not (rp-var-object 'c_3))(rp-var-define 'c_3 150000.0 'real #f))
(if (not (rp-var-object 'c_4))(rp-var-define 'c_4 5.0 'real #f))
(define  gui-hb-vis  ;;;; Create a panel for the user defined custom viscosity model
  (let ((panel #f)(CBH1)(CBH2)(CBM1)(CBM2))
     (define (update-cb . args);update panel fields
       (cx-set-real-entry CBH1 (rpgetvar 'c_1))
       (cx-set-real-entry CBH2 (rpgetvar 'c_2))
       (cx-set-real-entry CBM1 (rpgetvar 'c_3))
       (cx-set-real-entry CBM2 (rpgetvar 'c_4))    )
     (define (apply-cb . args)
       (rpsetvar 'c_1 (cx-show-real-entry CBH1))
       (rpsetvar 'c_2 (cx-show-real-entry CBH2))
       (rpsetvar 'c_3 (cx-show-real-entry CBM1))
       (rpsetvar 'c_4 (cx-show-real-entry CBM2))   )
     (lambda args
      (if (not panel)
       (let ((table) (form))
         (set! panel (cx-create-panel "Custom Viscosity(HB) Model" apply-cb update-cb))
         (set! table (cx-create-table panel "" 'border #f 'below 0 'right-of 0))
         (set! form (cx-create-frame table "" 'border #f))
         (set! CBH1 (cx-create-real-entry table "k" 'width 14 'row 1 'col 0 ))
         (set! CBH2 (cx-create-real-entry table "n" 'width 14 'row 2 'col 0 ))
         (set! CBM1 (cx-create-real-entry table "sigma_y" 'width 14 'row 3 'col 0 ))
         (set! CBM2 (cx-create-real-entry table "Critical_strain_mag" 'width 14 'row 4
   'col 0)))
          ) (cx-show-panel panel)    )))
(cx-add-item "User-Defined" "Viscosity Constants..." #\U #f cx-client? gui-hb-vis)
```



TUI for Patch

- The patch-function in fluent is available only through GUI
  - The following scheme function allows patching individual variables into zones
  - Usage: **`(tui-patch "X Velocity" '(2 3 9) '() 10.4)`**
    - Visit solve→initialize →patch panel to identify the variable name (e.g., 'X Velocity')
    - **`'(2 3 9)`** is a list of zone thread-ids; visit define→Boundary_condition panel for the ids
    - **`'()`** can contain any register-id - if you create them from adapt panel
    - **`10.4`** is the numeric value of the variable you want to patch

```
(define tui-patch
  (lambda (print-name t-id r value)
    (define (print-name->attribute-list name varlist)
      (let loop ((l varlist))
        (if (pair? l)
            (if (string=? name (cadar l))
                (car l)
                (loop (cdr l)))
            (cx-error-dialog "print-name->unknown variable name" name))))
    (let ((patch-var-list (%inquire-patch-variable-names))(attr))
      (set! attr (print-name->attribute-list print-name patch-var-list))
      (patch (list (list-ref attr 2) (list-ref attr 3) t-id r value)))))
```

Add Menu for Time Reset

- How can you reset 'Global / Physical time' in fluent?
  - The following scheme function does it - it allows the user to issue a command 'reset-time' from the Text-User-Interface (TUI)
  - The other straight forward scheme command to perform the same task would be **`(rpsetvar 'flow-time 0)`**; you may use any other appropriate value instead of **`'0'`**

```
;; this will show up in the solve/initialize TUI & GUI menu;
(define (reset-time)
  (let ((t-new (read-real "Global time" (rpgetvar 'flow-time))))
(format "\n Resetting global time to ~a" t-new)
(rpsetvar 'flow-time t-new)))
(ti-menu-insert-item!   initialize-menu    (make-menu-item "reset-time"
#t reset-time "Reset the global time."))
(if (and (cx-gui?) (not (symbol-bound? 'tr-defined? (the-environment))))
    (cx-add-item  "Initialize" "Reset-Time"  #\A #f
    (lambda () (and (cx-client?) (rp-unsteady?))) reset-time))
(define tr-defined? #t)
```