

Introduction to ANSYS Fluent scripting – Part 2 – Accessing solver data

Description	2	Benchmarking	26
Fluent Scheme procedures	3	Miscellaneous	26
Alias	3	Examples	27
Using text commands	4	Getting the path to the case file	27
Checking status of a case	5	Removing default interior zones from the list of post-processing surfaces	28
Accessing Fluent data	11	Changing type and settings of boundary conditions	29
Picking output from text commands	11	Creating multiple post-processing surfaces	29
Getting lists	13	Summary	32
Materials	13	Index	33
Domains / Phases	15	References	35
Threads / Zones	16		
Surfaces	20		
DPM	22		
Monitoring	25		

If you have questions regarding this document, please try to contact the author, first:
akram.radwan@ansys.com

Note that the content of this document is not covered by ANSYS support contracts. It is provided as self-help content for your convenience in addition to the contents of the ANSYS Fluent documentation. Always remember that ANSYS can decline to provide technical support if your project relies on Scheme scripts even if your problem description has nothing to do with Scheme.

Description

The Scheme programming language is mentioned in the ANSYS Fluent documentation and also in some self-help knowledge materials available on the ANSYS Customer Portal. But what are Scheme scripts and what is the difference to journal files? How to access Fluent cell zone or boundary conditions within a script? How to create user-defined menu items, keyboard shortcuts or panels?

These questions are answered in this three-part series.

The first document [1] describes Scheme. In principal you can find the content also in several digital books but not all of the commands and procedures described there can be used in ANSYS Fluent. The goal of this first document is to give you a basic understanding of the Scheme programming language. It is not a complete reference guide but you can write most of your scripts with what you learn here.

This second document [2] describes Fluent specific commands and procedures. You learn how to process text output and some convenience procedures. That document can only scratch the surface but it should give you the tools that you need to develop versatile scripts for ANSYS Fluent. You should be familiar with the contents of the first document before you start with the second part.

With the third document you can learn how to create your own menu items, keyboard shortcuts and panels. Most of this document is also covered in the official documentation [3] but the examples might make it easier to understand the concept. You should be familiar with the contents of the first document before you start with the third part. Knowledge about the second part is not required.

Although everything has been tested with Fluent 17.0, 17.1 and a prototype of 18.0, it is possible that you get unexpected results. Scheme is not documented and Fluent procedures and commands can change the behavior or stop working completely with every new minor release.

Important

Scheme is poorly documented and not covered by ANSYS support contracts. ANSYS can refuse to provide support if you have trouble with simulations that rely on Scheme scripts. ANSYS can also refuse to provide support for journal files that contain Scheme code.

Nevertheless, with Scheme you have a mighty tool at your disposal. You can automate simulation setup, post-processing and case modifications. You can even create your own menu items and panels to have quicker access to panels and settings you need most often.

Note: If you are interested in streamlining your processes in ANSYS Fluent you should also consider learning ACT (ANSYS Customization Toolkit). Starting with Fluent 17, ACT wizards can be used as beta features. There are demonstrations available in the ACT library of the ANSYS Customer Portal. ACT is not covered in this document.

Fluent Scheme procedures

There are literally hundreds of Scheme procedures built into Fluent. Most of them are not documented and can change any time. The list presented here are the most commonly used procedures. There might be more in the examples available on the ANSYS Customer Portal.

Important

You need to test Fluent Scheme procedures carefully for each new Fluent version. The behavior can change or they can stop working at all. There is no documentation available for them. The ANSYS technical support cannot assist you when you run into problems using one or several of the undocumented procedures available in this document or on the ANSYS Customer Portal.

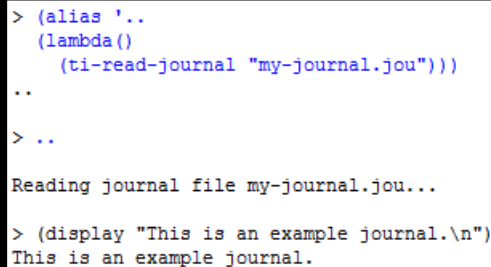
Important

If you cannot find what you need, you can contact ANSYS to get an offer for a custom-made script. Do not expect the ANSYS Technical Support to provide ready-to-use Scheme examples.

Alias

An alias is very similar to a named procedure. Essentially, you create a text command for a Scheme procedure.

001	(alias '..
002	(lambda ()
003	(ti-read-journal "my-journal.jou")))



```
> (alias '..
  (lambda()
    (ti-read-journal "my-journal.jou")))
..
> ..

Reading journal file my-journal.jou...

> (display "This is an example journal.\n")
This is an example journal.
```

- Line 1: Define the name “..” for the alias. It has to be given as symbol, not as string.
- Line 2: Start the procedure without parameters.
- Line 3: Read the journal “my-journal.jou” that is located in the Fluent working directory.

You can execute the procedure just by typing its name .. without parentheses.

If you want your aliases to be available in all new Fluent sessions, you can add its definition to the .fluent file.

Using text commands

At some point your scripts need access to Fluent text commands. While you can mix Scheme and text commands in journal files, you cannot do this inside a Scheme procedure. The text command needs to be passed to the Fluent console with `ti-menu-load-string` followed by the string of the text command.

```
(ti-menu-load-string "tui-command")
```

```
> (ti-menu-load-string "?exit")
?exitexit: Exit program.
#t
```

In this example the text command `?exit` shows the help text for the text command `exit`. There is no white space or line break between the output of the text command and its return value.

Note: You need to specify all arguments that are required by the text command.

There is a possibility to remove the output of text commands almost completely. You can route the output into a string with the procedure `with-output-to-string`. First, define a procedure with `with-output-to-string`. Then pass the text command to that procedure.

```
001 (define (quiet command)
002   (let ((eat-string))
003     (set! eat-string (with-output-to-string
004                       (lambda ()
005                         (ti-menu-load-string command))))))
```

```
> (define (quiet command)
  (let ((eat-string))
    (set! eat-string (with-output-to-string
                      (lambda ()
                        (ti-menu-load-string command))))))
quiet
```

Now you can use this new procedure to hide the output of most text commands. They are executed but most of the output and error messages are stored in a local variable that gets deleted afterwards.

```
> (ti-menu-load-string "/rep/si/awa inlet () mixture velocity-magnitude no")
/rep/si/awa inlet () mixture velocity-magnitude no
      mixture
      Area-Weighted Average
      Velocity Magnitude                      (m/s)
-----
                        inlet                  3.6111
#t

> (quiet "/rep/si/awa inlet () mixture velocity-magnitude no")
eat-string
```

Silent text commands can be useful if you don't want to spam your Fluent console with many commands that do something in the graphics window, for example during automatic post-processing with plots. When used inside a procedure, Fluent prints the name of the string variable only once after the procedure is finished.

```

> (define (mp1)
  (ti-menu-load-string "/dis/sm inlet ()")
  (ti-menu-load-string "/dis/sm outlet ()"))
mp1

> (mp1)
/dis/sm inlet ()/dis/sm outlet ()#t

> (define (mp2)
  (quiet "/dis/sm inlet ()")
  (quiet "/dis/sm outlet ()"))
mp2

> (mp2)
eat-string

```

Both example procedures (mp1) and (mp2) display the mesh of a zone called inlet and then of a zone called outlet. For (mp1) both text commands are printed in the Fluent console. However, for (mp2) only the specified string is printed once.

Note: It takes more time to execute text commands silently than to print it to the console. If your script requires a significant amount of time to run, consider printing everything to the console instead of hiding it.

It is also possible to pass on complete lists to text commands that support it. You need to use the format procedure that you know from the first part of Introduction to ANSYS Fluent scripting [1].

```

> (ti-menu-load-string (format #f "/dis/sm ~a" (list 'inlet 'outlet)))
/dis/sm (inlet outlet)#t

```

Checking status of a case

You have several possibilities to check in which state a simulation is.

You can check if a mesh or case is loaded with `case-valid?`. It returns the Boolean value `#t` if a mesh is available.

```

> (case-valid?)
#t

```

Because it returns a Boolean value, you can use it directly within an if statement.

```

> (if (case-valid?)
  (display "Mesh available")
  (display "Mesh NOT available"))
Mesh available

```

The same can be done for the data. `data-valid?` checks if the case is initialized. It does not differentiate if iterations were calculated already. It returns the Boolean value `#t` if cell values exist.

```

> (data-valid?)
#t

```

There is no Scheme command to check if the case is converged. This is stored as Boolean value in the RP variable solution/converged?

```
> (rpgetvar 'solution/converged?)  
#f
```

You can get the number of iterations with `client-inquire-iteration`. It is not necessary to initialize the case. For an uninitialized flow field, it reports zero iterations. The iteration is stored in the dat file. There is no RP variable that stores that value.

```
> (client-inquire-iteration)  
500
```

Note: If you want to reset the number of iterations to zero but keep the results, you can use the text command `/solve/monitors/residual/reset yes`. See also ANSYS solution 2004273 [4].

The same is true for `client-inquire-time-step` which gives you the current time step. Careful! The time step is also stored in the case file. Fluent reports the stored number also for steady state cases.

```
> (client-inquire-time-step)  
5
```

Alternatively, you can also access the rp var `time-step`.

```
> (rpgetvar 'time-step)  
5
```

Similarly, you can access the flow time with `client-inquire-flow-time` or the rp var `flow-time`.

```
> (client-inquire-flow-time)  
0.05  
  
> (rpgetvar 'flow-time)  
0.05
```

Note: If you want to reset the flow time to zero you can use `rpsetvar` to overwrite the current values.

There are many checks that you can do. These are useful if you need text commands that change their parameters with the models you use. If statements can be used together with one or multiple of the following procedures to invoke different text commands or to activate models.

You can also get an output of most of these variables with the text command `/file/show-configuration`.

General

(<code>rp-double?</code>)	Double #t or single precision #f
(<code>rp-3d?</code>)	3D #t or 2D case #f
(<code>rp-2d?</code>)	3D #f or 2D case #t
(<code>rp-axi?</code>)	2D axisymmetric or 2D axisymmetric swirl #t, 2D and 3D #f
(<code>client-solver-axi?</code>)	2D axisymmetric or 2D axisymmetric swirl #t, 2D and 3D #f

(sg-swirl?)	2D axisymmetrix swirl #t, everything else #f
(rp-seg?)	Pressure based solver #t, density based solver #f
(rp-unsteady?)	Transient #t or steady state simulation #f
(client-unsteady?)	Transient #t or steady state simulation #f
(rp-dual-time?)	Steady #f, transient first order 1, transient second order or bounded second order 2

Multiphase Models

(sg-mphase?)	Multiphase off #f, Volume of Fluid vof, Mixture drift-flux, Eulerian multi-fluid, Wet Steam #f
(client-has-multiple-domains?)	Multiphase off #f, Volume of Fluid vof, Mixture drift-flux, Eulerian multi-fluid, Wet Steam #f
(sg-wetsteam?)	Wet Steam active #t, not active #f
(sg-pb?)	Population balance add-on module loaded and active #t, #f if inactive or not loaded

Energy equation

(rf-energy?)	Energy enabled #t, disabled #f
--------------	--------------------------------

Viscous models / Turbulence models

(rp-lam?)	Laminar viscous model #t, all other turbulence models #f
(rp-turb?)	Turbulence model #t or laminar #f or inviscid flow used #f
(rp-inviscid?)	Inviscid #t, all other #f
(rp-sa?)	Spalart Allmaras 1, all other #f
(rp-ke?)	k-epsilon #t or any other turbulence model used #f
(rp-kw?)	k-omega #t or any other turbulence model used #f
(rp-kklw?)	Transition k-kl-omega #t, all other #f
(rp-trans-sst?)	Transition SST #t, all other #f
(rp-v2f?)	V2F #t, all other #f (V2F requires an additional license [5])
(sg-rsm?)	Reynolds Stress #t, all other #f
(rp-sas?)	Scale-Adaptive Simulation #t, all other #f
(rp-des?)	Detached Eddy Simulation #t, all other #f
(rp-les?)	Large Eddy Simulation #t, all other #f

Radiation models

(sg-rosseland?)	Rosseland radiation model #t, all other #f
(sg-p1?)	P1 #t, all other #f
(sg-dtrm?)	Discrete Transfer (DTRM) #t, all other #f
(sg-s2s?)	Surface to Surface (S2S) #t, all other #f
(sg-disco?)	Discrete Ordinate (DO) resolution (e.g. 1x1x1x1 → 8; 2x2x1x1 → 32), all other #f
(sg-montecarlo?)	Monte Carlo #t, all other #f
(sg-solar?)	Solar Loading / Solar Ray Tracing #t, disabled or DO Irridiation #f

Species transport and reactions

(rp-spe?)	Returns number of species for Species Transport, #f for combustion models
(rp-react?)	Species Transport with volumetric reactions #t, no volumetric reactions #f
(rp-spe-site?)	Returns number of site species, #f if wall surface reactions are disabled
(rp-spe-surf?)	Returns number of solid species, #f if wall surface reactions are disabled
(rp-spe-part?)	Returns number of solid species, #f if particle surface reactions are disabled
(rp-electro-chem?)	Electrochemical reactions active 1, #f if model is not active
(rp-potential?)	Electric potential active 1, #f if disabled, this is independent of the setting for electrochemical reactions
(sg-premixed?)	Premixed combustion #t, all other #f
(sg-par-premix?)	Partially Premixed Combustion #t, all other #f
(sg-pdf-transport?)	Returns number of volumetric species for Composition PDF Transport, all other #f
(sg-spark?)	Spark ignition active #t, not active #f. Note that it also returns #t after switching to a model that does not support ignition sparks if it was activated before.
(sg-ignite?)	Autoignition active #t, #f if disabled
(sg-nox?)	NOx model active #t, #f if disabled
(sg-sox?)	SOx model active #t, #f if disabled
(sg-soot?)	Soot model active #t, #f if disabled

Discrete phase model

(sg-dpm?)	Discrete phase interacts with continuous phase #t, #f if no interaction is allowed
-----------	--

Solidification and melting

(sg-melt?)	Solidification & Melting active #t, #f if inactive
------------	--

Acoustic models

(rp-acoustics?)	Ffowcs-Williams & Hawkings ffowcs, Broadband Noise Sources corre1, disabled #f
-----------------	--

Eulerian wallfilm

(rp-wallfilm?)	Eulerian Wall Film active #t, #f if disabled
(sg-wallfilm?)	Eulerian Wall Film active #t, #f if disabled

Dynamic mesh

(sg-dynmesh?)	Dynamic mesh enabled #t, #f if disabled, note that mesh motion (in cell zone conditions) does not count as dynamic mesh
---------------	---

User Defined

(sg-udm?)	At least one UDM (user-defined memory location) exists #t, #f if no UDM exists
(sg-node-udm?)	At least one node UDM exists #t, #f if no node UDM exists

([sg-uds?](#))

At least one UDS (user-defined scalar) exists #t, #f if no UDS exists

Solution Methods

([sg-noniterative?](#))

NITA (non-iterative time advancement) active #t, #f if disabled

Initialization

([hyb-init?](#))

Hybrid initialization is active #t, standard initialization is active #f

Note: This list is likely to be incomplete. You might find additional checks in different examples on the ANSYS Customer Portal. If you need to check for something that is not listed here, consider using text commands to activate or disable certain models which is probably faster and easier than experimenting to find a working Scheme procedure.

You can also get the absolute path and the filename of the case that is currently loaded.

([in-package cl-file-package rc-filename](#)) gives you the full path, including the file name without file extension. ([strip-directory \(in-package cl-file-package rc-filename\)](#)) gives you only the file name. All of these work only in combination with each other. See also ANSYS solution 2038952 [6].

```
> (in-package cl-file-package rc-filename)
E:/Ansys/Scheme/Scheme_Read_New_Profiles/SYS-1

> (strip-directory (in-package cl-file-package rc-filename))
SYS-1
```

Note: Fluent shows the path always with forward slashes as folder separators. This is also true on Windows systems, which use a back slash on a system level. If you need to pass the path to another application, you might need to convert the slash direction.

There is no simple way to strip the file name if you only need the path. But you can write a procedure that extract the path from the return value of the commands shown above. See chapter [Examples](#).

The file extension can't be extracted easily. You can check if a case was zipped with ([in-package cl-file-package comp-suffix](#)). This returns the string "gz" if the file was zipped and an empty string if it's not zipped.

For hd5/hdf files you can only check if a file on your hard drive has the type hd5 with ([hd-file? "file-name.cas.hd5"](#)) which returns a Boolean value. But you need to know the complete file name with the extension already.

You can also check the Fluent version with `client-inquire-version` and `client-inquire-release`.

([client-inquire-version](#))

Returns the version like it is shown in the title bar (e.g. 2d, dp, pbns, lam) as dotted pair

```
> (client-inquire-version)
(3d, dp, pbns, rke, transient . 3d, double precision, pressure-based, realizable k-epsilon, transient)
```

([client-inquire-release](#))

Returns the Fluent version number and the year of the release

```
> (client-inquire-release)
(17 0 0 2015)
```

(client-inquire-reference-depth)

Returns the reference depth for 2D planar simulations

```
> (client-inquire-reference-depth)
0.1
```

Reference Values	
Area (m2)	1
Density (kg/m3)	1.225
Depth (m)	0.1
Enthalpy (j/kg)	0
Length (m)	1
Pressure (pascal)	0
Temperature (k)	288.16
Velocity (m/s)	1
Viscosity (kg/m-s)	1.7894e-05
Ratio of Specific Heats	1.4

(%models-changed) / (dpm-parameters-changed)

These two commands are required if you modify RP variables that are required by active models. You can achieve the same if you open the panel or invoke a text command that updates the model where you changed the RP variable. The second command is only required if you change RP vars for DPM (discrete phase model).

(getenv "variable")

Returns the value of the environment variable passed to the procedure as string.

```
> (getenv "ansys170_dir")
C:\Program Files\ANSYS Inc\v170\ANSYS
```

(putenv "variable" "value")

Sets the value of an environment variable only for the current Fluent session. The value of other sessions is not changed.

```
> (putenv "ansys170_dir" "C:\Program Files")
ansys170_dir=C:\Program Files
```

(inquire-mesh-type)

Returns the mesh type. If the mesh contains more than a single element type it returns "mixed".

```
> (inquire-mesh-type)
quadrilateral
```

```
> (inquire-mesh-type)
hexahedral
```

```
> (inquire-mesh-type)
mixed
```

Accessing Fluent data

There are a number of procedures available to access zones, results or settings.

Picking output from text commands

The most important procedures are the pick procedures. See also ANSYS solutions 2039826 [7] and 2039638 [8] for information about pick.

To understand the examples, you can use the output of a text command like calculating a surface integral:

`/report/surface-integrals/area-weighted-avg`

```
> /rep/si/awa interior12 interior23 () temperature no
```

Area-Weighted Average Static Temperature		(k)
interior12	300.71983	
interior23	323.78086	
Net	312.25034	

`pick` grabs a string from the output of a text command. The actual output is not visible in the Fluent console.

```
> (pick "/rep/si/awa interior12 interior23 () temperature no")
312.25034

> (string? (pick "/rep/si/awa interior12 interior23 () temperature no"))
#t
```

By default, `pick` grabs the last output. Any number of white spaces separate the individual items. You can specify a number to pick a specific item from the output. Fluent counts backwards, starting with 1 and then going from right to left and from bottom to top.

```
> (pick "/rep/si/awa interior12 interior23 () temperature no" 1)
312.25034

> (pick "/rep/si/awa interior12 interior23 () temperature no" 2)
Net

> (pick "/rep/si/awa interior12 interior23 () temperature no" 3)
-----
```

```
> /rep/si/area-weighted-avg interior12 interior23 () temperature no
```

Area-Weighted Average Static Temperature		(k)
interior12	300.71983	
interior23	323.78086	
Net	312.25034	

If you want to see the output, you can use `pick-hard` instead of `pick`. The result is still a string.

```
> (pick-hard "/rep/si/awa interior12 interior23 () temperature no" 1)
/rep/si/awa interior12 interior23 () temperature no
      Area-Weighted Average
      Static Temperature                               (k)
-----
              interior12          300.71983
              interior23          323.78086
              -----
                        Net          312.25034
312.25034
```

To calculate with the result, you can either convert it to a number or you can use `pick-a-real`.

```
> (pick-a-real "/rep/si/awa interior12 interior23 () temperature no" 1)
312.25034

> (string? (pick-a-real "/rep/si/awa interior12 interior23 () temperature no" 1))
#f

> (number? (pick-a-real "/rep/si/awa interior12 interior23 () temperature no" 1))
#t
```

`pick-a-real` returns `#f` if the item you try to pick cannot be converted to a number.

```
> (pick-a-real "/rep/si/awa interior12 interior23 () temperature no" 2)
#f
```

Note: In this example the last valid item of the output is number 15. If you go further back, you get the text command itself. The empty parentheses cannot be captured. You get an error message if you go back too far.

```
> (pick "/rep/si/awa interior12 interior23 () temperature no" 15)
Area-Weighted

> (pick "/rep/si/awa interior12 interior23 () temperature no" 16)
no

> (pick "/rep/si/awa interior12 interior23 () temperature no" 20)
/rep/si/awa

> (pick "/rep/si/awa interior12 interior23 () temperature no" 21)
Error: CAR: invalid argument [1]: wrong type [not a pair]
Error Object: ()
```

With the `pick` procedures you can do a lot of scripting. Fluent can output listings of most of its settings with text commands. You can process this output one at a time with loops.

However, executing the same text command over and over again until you have everything you need can be very time consuming. It can be faster grabbing the complete output all at once and process the result. You can use `with-output-to-string`.

```

001 (define (my-capture-tui command)
002   (let ((my-string))
003     (set! my-string
004       (with-output-to-string
005         (lambda () (ti-menu-load-string command))))
006     (display my-string)))

```

```

> (my-capture-tui "/rep/si/awa interior12 interior23 () temperature no")
/rep/si/awa interior12 interior23 () temperature no
      Area-Weighted Average
      Static Temperature                               (k)
-----
            interior12                300.71983
            interior23                323.78086
-----
                        Net                312.25034

```

This example is just a stub. It shows you how to store the output of a text command in a string but not what to do with it. It's up to you to figure out how you can get what you need out of a single string like that.

Important

If you process such strings with your own procedures, you should compare the output of every new minor Fluent version with an older one. If the format of the output changes, your procedures might no longer work or produce incorrect results.

Getting lists

There are a number of procedures that provide useful lists. This is much more convenient than processing a large string and making your own procedure robust. However, there is no guarantee that the following procedures work in future versions of ANSYS Fluent.

Important

It is recommended to process the output of text commands whenever possible. If you use one or several of the following procedures, make sure to test the behavior thoroughly for each minor version of Fluent to avoid unexpected result.

Materials

([inquire-material-names](#))

Returns a list with all materials available in the case. It is similar to the text command /define/materials/list-materials. However, it does not list materials inside a mixture. The items in the list are symbols.

```

> (inquire-material-names)
(mixture-template anthracite air-piecewise air aluminum)

> /def/mat/lm
mixture-template
nitrogen(mixture-template)
oxygen(mixture-template)
water-vapor(mixture-template)
anthracite
air-piecewise
air
aluminum

```

`(inquire-material-names-all)`

Returns a list with all materials available in the case including the assignment of the materials as species in a mixture material.

```
> (inquire-material-names-all)
(mixture-template (nitrogen . mixture-template) (oxygen . mixture-template) (water-vapor . mixture-template) anthracite air-piecewise air aluminum)
```

`(get-all-materials)`

Returns a list with all materials and their properties. The list is large and can take some effort to process. The screenshot shows only the first few entries of the long list.

```
> (get-all-materials)
((anthracite inert-particle (chemical-formula . #f) (density (constant . 1550)) (specific-heat
```

`(get-material 'material-name)`

Returns a list with all properties of a specific material. For mixtures you also get a list of the names of the species.

```
> (get-material 'air)
(air fluid (chemical-formula . #f) (density (constant . 1.225)) (specific-heat (constant

> (get-material 'mixture-template)
(mixture-template mixture (chemical-formula . #f) (species (names h2o o2 n2)) (reactions
```

`(material-species-names (get-material 'mixture-name))`

Returns a list of the species. If the material name is not a mixture it returns #f.

```
> (material-species-names (get-material 'mixture-template))
(h2o o2 n2)

> (material-species-names (get-material 'air))
#f
```

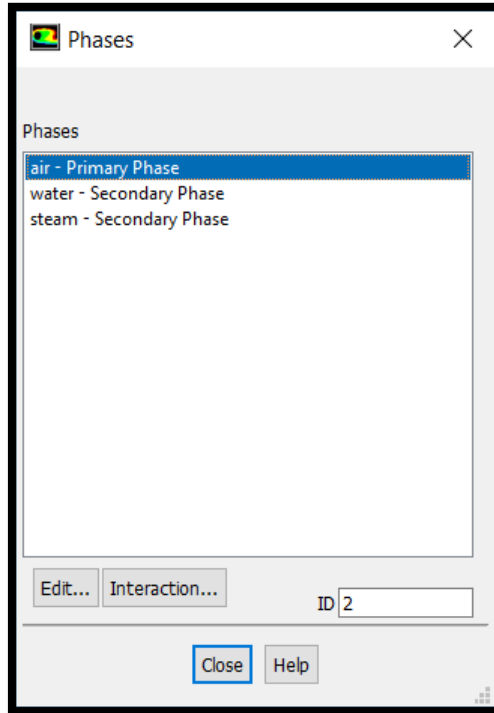
`(get-fluid-thread-material 'cell-zone-name)`

Returns all properties of the material available in a specific cell zone. For multiphase you get only the material of the primary phase.

```
> (get-fluid-thread-material 'part-solid)
(air-piecewise fluid (chemical-formula . #f) (density (incompressible-ideal-gas .
```

Domains / Phases

For the Fluent solver you can use the terms domains or phases interchangeably. For a single phase flow, you have one domain. For multiphase flows you have one domain for the mixture properties, one for the primary phase, one for each secondary phase and one for the interaction between the phases. It is very rare that you need access to interaction domains. Each domain is represented by a unique identifier that you can find in the phases panel. The mixture phase has always the id 1.



There is no text command to get a list of all domains but you can use several Scheme commands to get lists of the domains if you need it.

`(inquire-domain-names)`

Returns a list with all domain names as symbols.

```
> (inquire-domain-names)
(interaction steam water air mixture)
```

`(client-inquire-domain-ids-and-names)`

Returns a list with all domain names and their id as dotted pair inside the list.

```
> (client-inquire-domain-ids-and-names)
((mixture . 1) (air . 2) (water . 3) (steam . 4) (interaction . 5))
```

`(domain-name->id 'air)`

Returns the id of a domain name. You get an error message if the symbol is not a domain.

```
> (domain-name->id 'air)
2

> (domain-name->id 'wall13)

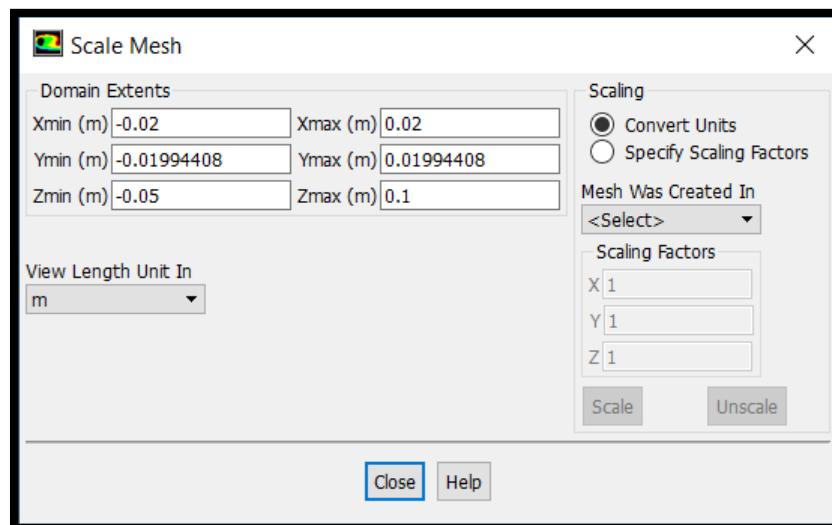
Error: CDR: invalid argument [1]: wrong type [not a pair]
Error Object: #f
```

There is no Scheme procedure available that returns the name from a known id.

([client-inquire-domain-extents](#))

Returns the dimensions of the complete computational domain as list. You get the same numbers from the scale panel.

```
> (client-inquire-domain-extents)
(-0.02 0.02 -0.019944076 0.019944076 -0.05 0.1)
```



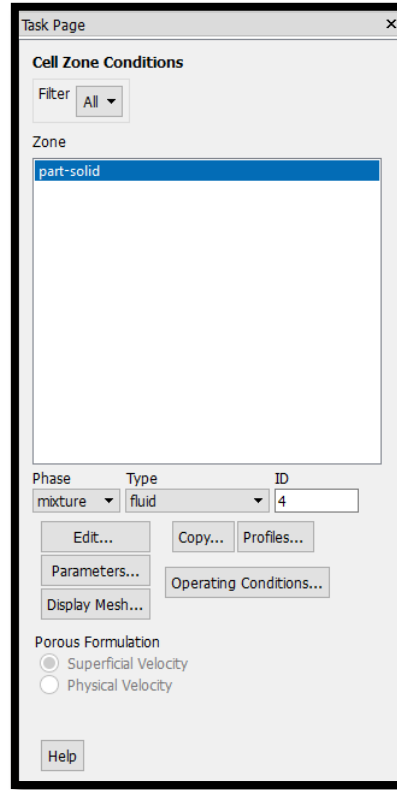
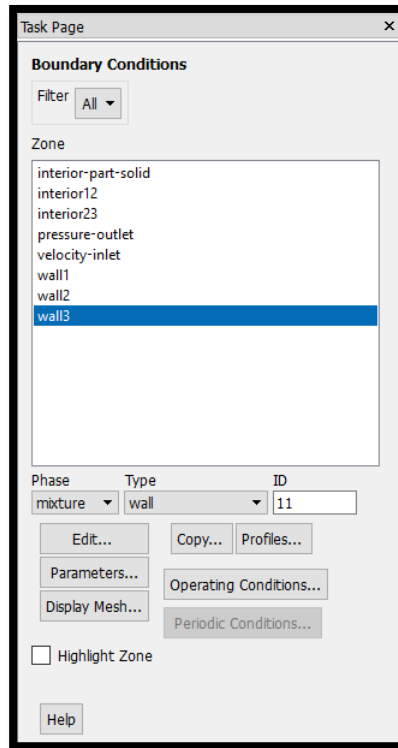
Threads / Zones

The terms threads and zones can be used interchangeably. These are cell or boundary zones. Typically, when you work with UDFs (user-defined functions) you call them threads, if you work in the GUI (graphical user interface) you call them zones.

You can't create new zones without touching the mesh. If you need different zones you have to go back to you mesh generator or split, slice or merge your existing zones in a serial Fluent session.

Each zone has a type associated with it. The type determines how the solver treats a zone. A special type is the so-called default-interior. It consists of all the cell faces of the volume mesh that are not associated with another named face zone. Typically, you do not need access to these default-interior zones but the solver needs them. See ANSYS solution 2041732 [9] for more details about default-interior zones.

Each zone has a unique identifier that you can find on the cell zones conditions and boundary conditions task pages.



`(client-inquire-zone-names) / (inquire-zone-names) / (inquire-thread-names)`

Returns a list with all available zone names including cell zones, boundary conditions and default-interior zones. Post-processing surfaces are not in the list.

```
> (client-inquire-zone-names)
(part-solid interior-part-solid interior12 interior23 velocity-inlet pressure-outlet wall1 wall2 wall3)
```

`(inquire-cell-thread-names)`

Returns a list with all available names of cell zones. Fluent returns the names as symbols.

```
> (inquire-cell-thread-names)
(part-solid)
```

`(client-inquire-cellzone-types)`

Returns a list with all cell zone types that Fluent knows. The list has nothing to do with the cell zones that are available in the current case.

```
> (client-inquire-cellzone-types)
(fluid solid)
```

`(inquire-cell-thread-ids)`

Returns a list with the ids of the cell zones. The order should be identical with the order of `inquire-cell-thread-names`.

```
> (inquire-cell-thread-ids)
(4)
```

`(inquire-face-thread-names)`

Returns a list with all available names of face zones, including boundaries, interiors and default-interiors. Fluent returns the names as symbols.

```
> (inquire-face-thread-names)
(interior-part-solid interior12 interior23 velocity-inlet pressure-outlet wall1 wall2 wall3)
```

`(inquire-boundary-thread-names)` / `(client-inquire-boundary-zones)`

Returns a list with all available names of boundary zones, including fans, porous jump and radiators. Interiors and default interiors are not part of the list. Fluent returns the names as symbols.

```
> (inquire-boundary-thread-names)
(velocity-inlet pressure-outlet wall1 wall2 wall3)
```

`(client-inquire-interior-zones)`

Returns a list with all available names of interior and default-interior zones.

```
> (client-inquire-interior-zones)
(interior-part-solid interior12 interior23)
```

`(client-inquire-facezone-types)` / `(client-inquire-zone-types)`

Returns a list with all face zone types that Fluent knows. The list has nothing to do with the face zones that are available in the current case.

```
> (client-inquire-facezone-types)
(axis degassing exhaust-fan inlet-vent intake-fan interface interior mass-f
```

`(inquire-face-thread-ids)`

Returns a list with the ids of all face zones. The order should be identical with the order of `client-inquire-zone-names`.

```
> (inquire-face-thread-ids)
(1 2 3 7 8 9 10 11)
```

`(inquire-face-zone-name-type-list)`

Returns a list with all available face zone names and their type. Fluent makes no difference between interior and default-interior zones.

```
> (inquire-face-zone-name-type-list)
((thread-type) (interior-part-solid interior) (interior12 interior) (interior23 interior) (v
```

`(inquire-thread-names-of-type 'type) / (client-inquire-zones-of-type 'type)`

Returns a list of cell or face zone names that match the given type. It returns an empty list if no match exists.

```
> (client-inquire-zones-of-type 'wall)
(wall1 wall2 wall3)

> (client-inquire-zones-of-type 'fluid)
(part-solid)

> (client-inquire-zones-of-type 'solid)
()
```

`(thread-name->id "name") / (thread-name->id 'name) / (client-zone-id "name") / (client-zone-id 'name)`

Returns the thread id of a given thread or zone name. You can pass the name as string or as symbol. It works for cell and face zones.

```
> (thread-name->id "wall3")
11

> (thread-name->id 'wall3)
11
```

`(thread-id->name id) / (client-zone-id->name id)`

Returns the name as symbol of a given zone id. It works for cell and face zones.

```
> (thread-id->name 11)
wall3
```

`(%is-default-interior? (thread-name->id 'interior-part-solid))`

```
> (%is-default-interior? (thread-name->id "interior-part-solid"))
#t

> (%is-default-interior? (thread-name->id "part-solid"))
#f
```

`(get-thread-list "name-pattern")`

Returns a list of all thread names (= zone names and boundary names) that match the name pattern. You can use * as wild card. The names in the list are symbols, not strings!

```
> (get-thread-list "velocity")
()

> (get-thread-list "velocity*")
(velocity-inlet)

> (get-thread-list "*city*")
(velocity-inlet)

> (get-thread-list "part*")
(part-solid)

> (get-thread-list "*part*")
(interior-part-solid part-solid)
```

`(inquire-adjacent-threads id) / (%inquire-adjacent-threads id)`

The id has to be a face zone id or Fluent returns an error. For face zones Fluent returns a list with four members:

1. Cell zone id on the first side of a face zone.
2. Cell zone id on the second side. This is #f for all boundary zones that have no boundary on the opposite side.
3. Face zone id of the shadow zone. Shadow zones exist only for baffles. This is #f for all non-baffle face zones.
4. The meaning of the last item of the list is unknown.

```
> (%inquire-adjacent-threads 2)
(4 4 #f #t)

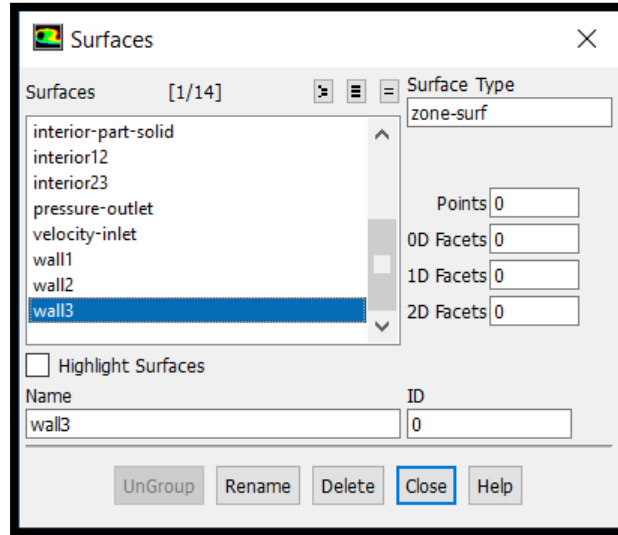
> (%inquire-adjacent-threads 4)

Error: inquire-adjacent-threads: not a face thread
Error: inquire-adjacent-threads: not a face thread
Error Object: #f
```

Surfaces

Fluent uses a different data structure for post-processing than for the solver itself. Unlike cell and boundary zones, post-processing surfaces don't have to be aligned with the mesh. Fluent creates a surface for each zone when you read in a mesh for the first time. However, it does not create the surfaces automatically if you read a new mesh over an existing case. Furthermore, you can't access surfaces from UDFs.

Like zones, surfaces have unique identifiers. Since the data structure is different, the ids can be different even for the surfaces Fluent creates automatically. You can find them in the surface manage panel.



`(inquire-surface-names)`

Returns a list with all available post-processing surface names. Unfortunately, Fluent creates post-processing surfaces also for default-interiors. Using default-interiors for post-processing can potentially terminate your Fluent session because these zones consist of a huge number of cell faces. Since the names of zones and default surfaces are identical, you can work around this problem easily. You can find an example later.

```
> (inquire-surface-names)
(wall3 wall2 wall1 pressure-outlet velocity-inlet interior23 interior12 interior-part-solid _x=0 _y=0 _z=0 _point_origin _line_center _plane_diagonal)
```

You can also list only point, line or plane surfaces. There is no procedure to list iso surfaces, though.

`(inquire-point-surface-names)`

```
> (inquire-point-surface-names)
(_point_origin)
```

`(inquire-surface-line-names)`

```
> (inquire-surface-line-names)
(_line_center)
```

(`inquire-surface-plane-names`)

```
> (inquire-surface-plane-names)
(_plane_diagonal)
```

(`inquire-surface-ids`)

Returns a list of all surface ids. The order should be identical to `inquire-surface-names`.

```
> (inquire-surface-ids)
(0 1 2 3 4 5 6 7 8 9 10 11 12 13)
```

(`surface-id->name` id)

Returns the surface name for a known surface id. Note that surface ids and zone ids are different.

```
> (surface-id->name 0)
wall3
```

(`surface-name->id` 'name)

Returns the surface id for a known surface name. Note that surface names and zone names can be identical.

```
> (surface-name->id 'wall3)
0
```

(`delete-surfaces` '(ids))

Deletes a list of surfaces. This is similar to the text command `/display/surface/delete-surface`. The text command can only delete one surface at a time while the Scheme command can delete many surfaces at once.

```
> (delete-surfaces '(10 11 12 13))
(#[primitive-procedure cx-delete-list-items] (_z=0 _point_origin _line_center _plane_diagonal))
```

(`client-all-symmetry-planes`)

Returns a list of all boundary zones of type symmetry.

```
> (client-all-symmetry-planes)
(symmetry_top)
```

DPM

(`inquire-injection-names`)

Returns a list with all DPM (discrete phase model) injections currently defined.

```
> (inquire-injection-names)
(injection-0 injection-1 injection-2 injection-side-0 injection-side-1)
```

(dpm-inquire-particle-types)

Returns a list of all possible particle types with the models that are active at the moment.

```
> (dpm-inquire-particle-types)
(inert massless)
```

(dpm-clear-all-particles)

Removes all particles from the simulation. This is identical with the text command `/define/models/dpm/clear-particles-from-domain` which is available for transient particle tracking.

(dpm-summary)

Prints the summary of the last particle tracking. This is identical with the text command `/report/dpm-summary`

```
> (dpm-summary)

Fate                Number      Elapsed Time (s)      Injection, Index
      Min      Max      Avg      Std Dev      Min      Max
----
Incomplete          4      1.202e+00  1.202e+00  1.202e+00  0.000e+00  injection-0  0  injection-0  0

(*)- Mass Transfer Summary -(*)

Fate                Mass Flow (kg/s)
      Initial      Final      Change
----
Incomplete          4.000e-20  4.000e-20 -6.019e-36

(*)- Energy Transfer Summary -(*)

Fate                Heat Rate (W)
      Initial      Final      Change
----
Incomplete          1.243e-16  1.243e-16 -2.465e-32
```

There are three procedures that go beyond a dpm-summary. Essentially they give you integral results of a dpm sampling. Usually they are used for transient dpm monitoring. They go beyond the reporting capabilities available in the Fluent GUI.

(dpm-one-inj-summary-by-inj-name-and-fate-and-quantity 'inj-name 'fate 'quantity)

(dpm-all-injj-summary-by-fate-and-quantity 'fate 'quantity)

(dpm-all-injj-summary-all-evaporated-mass-by-quantity-suffix 'quantity)

'inj-name is the name of an injection as symbol or string.

'fate is the fate of the particle as symbol. Strings cannot be used here. You can use also a boundary zone id to get the escaped particles through a specific zone.

Ten fates are available. They can even be used if the model (e.g. DPM wall film) is not active but the output will be zero.

- 'aborted
- 'incomplete
- 'evaporated
- 'absorbed
- 'influid
- 'infilmm
- 'deleted
- 'escaped
- 'trapped

'quantity' is the parameter you're interested in. 22 quantities are available. Like fates, you can use a quantity even if the required model is not active.

- 'number
- 'average-time
- 'min-time
- 'max-time
- 'injection-min
- 'injection-max
- 'exit-mass
- 'initial-mass
- 'exit-enthalpy
- 'exit-sensible-enthalpy
- 'initial-sensible-enthalpy
- 'latent-enthalpy
- 'reaction-enthalpy
- 'exit-char-mass
- 'initial-char-mass
- 'exit-volatile-mass
- 'initial-volatile-mass
- 'exit-liquid-mass
- 'initial-liquid-mass
- 'exit-species-mass-...
- 'initial-species-mass-...

Note that these procedures report integral values from the beginning to the current time step for most of the combinations of quantities and fates. If you are interested only in the current time step you need to store the value from the previous time step in a variable and subtract them.

Examples:

```
(dpm-one-inj-summary-by-inj-name-and-fate-and-quantity 'injection-0 'influid 'number)
```

Reports the number of particles of injection-0 that are currently in the system.

```
(dpm-one-inj-summary-by-inj-name-and-fate-and-quantity 'injection-0 8 'number)
```

Reports the number of particles of injection-0 that passed through boundary zone 8 since the start of the simulation.

```
(dpm-all-injj-summary-by-fate-and-quantity 'escaped 'exit-mass)
```

Reports the mass of all particles of all injections that escaped through any boundary since the start of the simulation.

Note: You might need to set an RP var before you can use zone-specific reporting like in the second example.

```
(rpsetvar 'dpm/per-inj-thread-summaries? #t)
```

Important

If you want to use these procedures use a very simple test case and experiment with the combinations of fates and quantities. Not all combinations give reasonable output therefore it is very important to test it, first.

You can use these procedures together with Scheme custom monitors.

Monitoring

It is possible to access the monitoring capabilities of Fluent with plot-custom-value.

Important

You can think of monitors as legacy feature in Fluent. Reports are more versatile and might replace monitors in the future. In Fluent 17 you can use the procedure plot-custom-value like it is described here but it is possible that it does not work that way in future versions.

```
(plot-custom-value (procedure) "file-name" window-number boolean-delete boolean-transient  
"monitor-name")
```

1. (procedure) is the name of a procedure that returns a single number that can be monitored.
2. "file-name" is the name of the file the results are written to. Fluent creates two files, one with the name like you have written it and one with the file extension *.xy. The second file is the final output; the first one contains temporary data. This is required, you can't just plot the results without writing them to a file.
3. window-number is the number of the window you want to plot the monitor to. If you do not want to plot, use #f instead.
4. boolean-delete is a Boolean value that tells Fluent if the file should be retained (#f) or deleted (#t). This should *always* be #f.
5. boolean-transient tells Fluent if it should plot your value over iterations (#f) or the flow-time (#t). You can't plot over the time step.
6. "monitor-name" is the name of the quantity, printed left of the y-axis and in the header of the output file.

You need to execute the command in each iteration or time-step either by calling it from a script or by adding it to Execute Commands (Tree: Solution > Calculation Activities > Execute Commands).

Example:

```
(plot-custom-value (dpm-all-injj-summary-by-fate-and-quantity 'escaped 'exit-mass) "dpm-  
exit" 3 #f #t "DPM Exit Mass")
```

Monitors the total mass of the escaped particles since the beginning of the simulation. The result is written to a file in the xy file format with the name dpm-exit.xy and it is plotted in window 3. The data is appended to the file and it uses the flow time for the x axis. The name of the y axis is DPM Exit Mass.

Benchmarking

If you want to know how long Fluent takes to execute a Scheme procedure, you can use `benchmark` to measure the time. For more information, see ANSYS solution 2014956 [10].

```
> (benchmark '(ti-menu-load-string "def bc list"))
def bc list id name type material kind
-----
4 part-solid fluid luft_vdi cell
1 interior-part-solid interior face
2 interior12 interior face
3 interior23 interior face
7 velocity-inlet velocity-inlet face
8 pressure-outlet pressure-outlet face
9 wall1 wall luft_vdi face
10 wall2 wall luft_vdi face
11 wall3 wall luft_vdi face
cpu-time: cortex=0.02294495661971041, solver=0
elapsed-time: 0
#t
```

Miscellaneous

`(%udf-on-demand 'macro_name::library_name)`

You can load UDFs of type `DEFINE_ON_DEMAND` with `%udf-on-demand`. This is identical to the text command `/define/user-defined/execute-on-demand`. The parameter has to be passed as symbol.

It is not possible to choose the color of mesh lines with text commands. Even `update-scene` does not allow to change line colors. This is only possible with the scene description panel in the GUI.

You can change all line colors at once with the combination of a Scheme procedure and a text command:

```
(cx-store-surface-all-attr 'edge-color '(0.4 0.6 0.8))
/display/re-render
```

The Scheme command sets the red, green and blue values of the edge color. The values range from 0 to 1. The text command triggers an update of the view.

If you want several zones in different colors you can use the overlay mode and display the different parts of the mesh one after another.

`(cx-interrupt)`

You can use `cx-interrupt` to interrupt a simulation within a Scheme procedure. This is similar to pressing `CTRL + C` on your keyboard during a simulation. This can be useful if you want to define your own convergence criteria or if you need to stop the simulation before it diverges or calculates unphysical values. Typically, you use this procedure within an `if` statement.

Examples

Some of these examples are also available as individual solutions on the ANSYS Customer Portal. Of course, this list is not complete. To find more examples on the Customer Portal search the solution database for the keyword “scm” and filter for the product ANSYS Fluent.

If you want to learn how similar versions of these examples could be developed, see the attached document 2042682_Detailed_Examples.pdf.

Getting the path to the case file

Sometimes it can be good to get the path or the case name. This example shows how you get the full path name without the case. If you are only interested in the case name you can also take a look at ANSYS solution 2038952 [6].

```
001 (define (my-get-path)
002   (let ((my-full-path (in-package cl-file-package rc-filename))
003         (my-path)
004         (my-pathlength)
005         (my-last-folder 0))
006     (set! my-pathlength (string-length my-full-path))
007     (do ((i 0 (+ i 1))) ((eqv? i my-pathlength))
008       (if (char=? #\ / (string-ref my-full-path i))
009         (set! my-last-folder i)))
010     (set! my-path (substring my-full-path 0 my-last-folder))
011     (display my-path)))
```

```
> (my-get-path)
E:/Ansys/Scheme/Scheme_Read_New_Profiles
```

- Line 1: Define the name “my-get-path” for the procedure. No arguments required.
- Lines 2 - 5: Define local variables.
- Line 2: The variable “my-full-path” holds the full path including the file name without file extension
- Line 6: Store the total length of the string in the variable “my-pathlength”. This is required to abort the loop in the next line.
- Line 7 – 9: Loop through each character of the path. The iteration variable “i” starts at zero and is increased by one in each loop. If it is identical with the length of the path string the loop stops. Note that the last character of the variable “my-full-path” will not be processed because of the eqv? check. This is not required in this case because the last character is part of the file name and not the path itself.
- Line 8: Check if the current character of the path is identical with the forward slash character. Nothing happens if it is not identical and the loop continues with the next character.
- Line 9: Set the index of the folder separation character. This is updated for each new folder. After all individual characters of the path are checked, the variable “my-last-folder” holds the number of characters that belong to the path, excluding the last “/”.
- Line 10: Take all characters from the string “my-full-path” from 0 to “my-last-folder” and store the result in the variable “my-path”.
- Line 11: Print the result to the Fluent console.

Removing default interior zones from the list of post-processing surfaces

There is not much you can do with default interior zones from a post-processing perspective. Nevertheless, Fluent creates them automatically from the boundary zone. You can use the following Scheme script to remove all default interior zones. Of course, they stay in the boundary zones list but they do not get in the way during post-processing. See also solution 2042751 [11].

```
001 (define (remove-default-interior)
002   (let ((default-interior-list '()))
003     (for-each (lambda (zone-name)
004       (if (%is-default-interior? (thread-name->id zone-name))
005         (if (not (eqv? #f (member zone-name (inquire-surface-names))))
006           (set! default-interior-list (append default-interior-list (list
007             (surface-name->id zone-name))))))
008       (inquire-face-thread-names))
009     (if (eqv? default-interior-list '())
010       (display "No default interior zones left as surface\n")
011       (begin
012         (display (format #f "The following surfaces are identified as
013         default interior zones\n~a" (map surface-id->name default-interior-list)))
014         (newline)
015         (delete-surfaces default-interior-list))))))
```

- Line 1: Define the name “remove-default-interior” for the user-defined Scheme procedure.
- Line 2: Define the variable “default-interior-list” as empty list.
- Line 3, 8: Loop through all boundary zones. The name of the current zone is stored in the variable “zone-name”.
- Line 4: Check if the current zone is a default interior zone. Fluent executes the following code only if this condition returns #t. If it is #f Fluent continues with the next boundary.
- Line 5: Check if a surface exists that has the same name as the default interior boundary. The statement looks complicated because of the return value of the procedure member. The inner parentheses returns #f if the zone name is not found in the list of surface names but it returns the tail of the list if it does exist. The parentheses starting with eqv? returns #t if the surface name does not exist and #f if it exists. This return value is negated that the if statement gets the value #t if the default interior zone name also exists as surface.
- Line 6: Add the id of the current surface to the list to delete them later. The inner parentheses converts the name of the surface to the surface id which is different to the zone id used in line 4. This id is put into a list with only one member. This is necessary to append it to the existing list stored in the variable “default-interior-list”.
- Line 8: Check if the variable “default-interior-list” is only an empty list. If it is empty, go to line 9. If it is not empty, go to the block that starts in line 10.
- Line 9: Report that there are no surfaces for default interior zones. The script stops here.
- Line 11: Display the names of the surfaces that are deleted. The inner parentheses converts the surface id back to readable names.
- Line 13: Delete all surfaces that are stored in the list “default-interior-list”.

Changing type and settings of boundary conditions

In general, it is recommended to set the type of boundary conditions during preprocessing. But if you need to adjust many boundary conditions at the same time, you can adjust this example to your needs. See also ANSYS solution 2039572 [12]. The text command used in this example works only for laminar flow.

```
001 (define (change-zone-wall-hflux zone-prefix hflux)
002   (if (and (string? zone-prefix) (number? hflux))
003     (let ((zone-list))
004       (set! zone-list (get-thread-list (string-append zone-prefix "*")))
005       (for-each (lambda (zone)
006         (ti-menu-load-string (string-append "/define/boundary-
007 conditions/zone-type " (symbol->string zone) " wall\n"))
008         (ti-menu-load-string (format #f "/define/boundary-
009 conditions/wall ~a 0 no 0 no no no ~a no no no no 1\n" (symbol->string
zone) hflux)))
zone-list))))
```

- Line 1: Define the name “change-zone-wall-hflux” of the procedure and two parameters. The first one must be a string, the second one a number.
- Line 2: Check if both parameters have the correct type. Do nothing if they are incorrect.
- Line 3: Define a local variable.
- Line 4: Store a list of all zones that start with the provided string in the variable “zone-list”.
- Line 5 to 9: Loop through the list of zones. The variable “zone” holds the name of the current zone as symbol. The conversion of the symbol to a string could also be done with format instead of a combination of string-append and symbol->string.
- Line 6: Set the current zone to the type “wall”.
- Line 8: Set the boundary conditions for the current zone.

Creating multiple post-processing surfaces

If you need many post-processing surfaces, you can adjust the following script to your needs. See also ANSYS solution 2042772 [13] for an alternate version with a panel (discussed in part 3, ANSYS solution 2042683 [2]). Solutions 912 [14] and 1030 [15] can also be of interest to get additional ideas how to adjust the script for different applications.

```
001 (define (post-iso-n-domain m-number m-direction m-name-prefix)
002   (if (and (data-valid?) (integer? m-number) (> m-number 0) (or (eqv? m-
direction 0) (eqv? m-direction 1) (eqv? m-direction 2) (string-ci=? m-
direction "x") (string-ci=? m-direction "y") (string-ci=? m-direction "z")))
(string? m-name-prefix))
003   (begin
004     (let ((m-iso) (m-name) (m-pos) (m-distance) (m-min) (m-max))
005       (cond
006         ((or (eqv? 0 m-direction) (if (string? m-direction) (string-ci=?
"x" m-direction)))
007           (begin
008             (set! m-iso "x-coordinate")
009             (set! m-min (list-ref (client-inquire-domain-extents) 0))
010             (set! m-max (list-ref (client-inquire-domain-extents) 1))))
011         ((or (eqv? 1 m-direction) (if (string? m-direction) (string-ci=?
"y" m-direction)))
```

```

012         (begin
013             (set! m-iso "y-coordinate")
014             (set! m-min (list-ref (client-inquire-domain-extents) 2))
015             (set! m-max (list-ref (client-inquire-domain-extents) 3))))
016         ((or (eqv? 2 m-direction) (if (string? m-direction) (string-ci=?
"z" m-direction))))
017         (begin
018             (set! m-iso "z-coordinate")
019             (set! m-min (list-ref (client-inquire-domain-extents) 4))
020             (set! m-max (list-ref (client-inquire-domain-extents) 5))))
021         (display m-iso)(display " ")
022         (set! m-distance (/ (+ (abs m-min) (abs m-max)) (+ m-number 1)))
023         (display m-distance)(newline)
024         (do ((i 1 (+ i 1)))
025             ((> i m-number) (display (format #f "\nCreated ~a iso-surfaces
along ~a.\n" m-number m-iso))))
026         (begin
027             (set! m-pos (+ m-min (* i m-distance)))
028             (display m-pos)(display " ")
029             (set! m-name (format #f "_~a-~a" m-name-prefix i))
030             (if (member (string->symbol m-name) (inquire-surface-names))
031                 (delete-surfaces (list (surface-name->id (string->symbol m-
name))))))
032             (display m-name)(newline)
033             (ti-menu-load-string (format #f "/surface/iso-surface ~a ~a ()
() ~a ()\n" m-iso m-name m-pos))))))
034         (display "Error: Invalid input\n"))

```

- Line 1: Define the name "post-iso-n-domain" of the procedure with three parameters.
- Line 2: Do some sanity checks for the input parameters. The error in line 34 is shown if one of them fails.
Check if the data is valid. Iso-surfaces cannot be created if no data is available.
Check if the first parameter is an integer that is larger than zero.
Check if the second parameter is either the number 0, 1, 2 or the string x, X, y, Y, z or Z. Note that it does not check if the case is 2d or 3d which might be good to add to avoid errors for 2d cases.
Check if the last parameter is a string.
- Line 3: Start the code block that executes when the sanity checks of line 2 are all passed. It ends in line 33.
- Line 4: Define six local variables
- Line 5: Start a condition block that ends in line 20.
- Line 6: Check if the planes align on the x-axis (input is either 0, x or X).
- Line 8: Set the variable "m-iso" to the string "x-coordinate". This is used in line 33 to define the parameter for the iso-surface.
- Line 9, 10: Get the min and max domain extents in the x direction.
- Line 11-20: Like line 6-10 but for y and z directions.
- Line 21, 23, 28, 32: Show the values of some variables. Such outputs can be very useful during the development of a script but should be either removed or commented out when you are finished to avoid unnecessary clutter of the console which takes time.
- Line 22: Calculate the distance between the iso-surfaces.
- Line 24: Start a loop to create the iso-surfaces
- Line 25: Stop the loop after all planes are created and display a success message.
- Line 27: Calculate the position as global coordinate for the current iso-surface.

- Line 29: Derive the name of the plane from a constant string, an input parameter and the current iteration number of the loop.
- Line 30, 31: Check if a surface with the name from line 29 exists already and delete it if necessary.
- Line 33: Create the iso-surface with a text command.
- Line 34: Error message in case the sanity checks in line 2 fail.

Always remember that text commands are not robust. The number of their parameters can depend on the models you use. For example, for Eulerian multiphase simulations you would need one additional parameter for the phase. You can add tests to make the script more robust. For example:

033	(if (eqv? #f (client-has-multiple-domains?))
034	(ti-menu-load-string (format #f "/surface/iso-surface ~a ~a
	() () ~a ()\n" m-iso m-name m-pos))
035	(ti-menu-load-string (format #f "/surface/iso-surface mixture
	~a ~a () () ~a ()\n" m-iso m-name m-pos))))))
036	(display "Error: Invalid input\n"))

However, it can be very difficult to think about all possible model combinations that might change a text command in advance. If you work with new models, test your script with small test examples, first. This is very important to ensure the robustness.

Summary

Scheme offers you a versatile toolset to automate tasks in ANSYS Fluent. The most important tool when working with results or input data is the pick procedure. You can grab almost everything that Fluent prints to its console with it.

A number of other procedures allow you easy access to lists of zones, surfaces or materials. Although they can be handy, you need to keep in mind that none of these procedures are officially supported. They could stop working with a new minor release of ANSYS Fluent.

If you have trouble with your Fluent simulations, rename the .fluent file and try to reproduce the issues without using scripts.

Contact the ANSYS technical support only when you are sure that the problem exists without using Scheme.

If you need a *reliable* customization or scripting solution for your projects for a specific Fluent version, you can contact the ANSYS consulting team. Provide a detailed description of your requirements and ANSYS can make you an offer to develop a solution for you.

This document is provided for your convenience as self-help content. No support is provided for its content. The examples were tested with Fluent 17.0, 17.1 and a development version of Fluent 18. But ANSYS cannot guarantee that everything works with these or any other releases.

If you are interested to learn more about Scheme basics, you can use the first part of the Introduction to Fluent scripting. You can find it in ANSYS solution 2042681 [1].

If you want easy access to your scripts with your own menu items, panels and keyboard shortcuts, check out part three of Introduction to Fluent scripting. You can find it in ANSYS solution 2042683 [2].

Index

%	
%inquire-adjacent-threads.....	20
%is-default-interior?.....	19
%-is-default-interior?	28
%udf-on-demand	26
A	
alias	3
C	
case-valid?.....	5
cl-file-package.....	9, 27
client-all-symmetry-planes	22
client-has-multiple-domains?	7, 31
client-inquire-boundary-zones	18
client-inquire-cellzone-types	17
client-inquire-domain-extents	16, 30
client-inquire-domain-ids-and-names	15
client-inquire-facezone-types.....	18
client-inquire-flow-time.....	6
client-inquire-interior-zones	18
client-inquire-iteration	6
client-inquire-reference-depth.....	10
client-inquire-release	9
client-inquire-time-step	6
client-inquire-version	9
client-inquire-zone-names	17
client-inquire-zones-of-type	19
client-inquire-zone-types	18
client-solver-axi?.....	6
client-unsteady?	7
client-zone-id	19
client-zone-id->name	19
cx-interrupt.....	26
cx-store-surface-all-attr.....	26
D	
data-valid?	5
delete-surfaces	22, 28
domain-name->id.....	15
dpm-all-injj-summary-all-evaporated-mass-by- quantity-suffix	23
dpm-all-injj-summary-by-fate-and-quantity	23, 24, 25
dpm-clear-all-particles	23
dpm-inquire-particle-types	23
dpm-one-inj-summary-by-inj-name-and-fate-and- quantity.....	23, 24
dpm-parameters-changed	10
dpm-summary.....	23
G	
get-all-materials	14

getenv	10
get-fluid-thread-material.....	14
get-material	14
get-thread-list	20, 29

H

hyb-init?	9
-----------------	---

I

in-package	9, 27
inquire-adjacent-threads	20
inquire-boundary-thread-names	18
inquire-cell-thread-ids	18
inquire-cell-thread-names	17
inquire-domain-names	15
inquire-face-thread-ids	18
inquire-face-thread-names.....	18, 28
inquire-face-zone-name-type-list	18
inquire-injection-names.....	22
inquire-material-names	13
inquire-material-names-all	14
inquire-mesh-type	10
inquire-point-surface-names	21
inquire-surface-ids	22
inquire-surface-line-names	21
inquire-surface-names	21, 28, 31
inquire-surface-plane-names	22
inquire-thread-names.....	17
inquire-thread-names-of-type	19
inquire-zone-names	17

M

material-species-names.....	14
models-changed	10

P

pick.....	11
pick-a-real	12
pick-hard	12
plot-custom-value.....	25
putenv	10

R

rc-filename	9, 27
rf-energy?.....	7
rp-2d?.....	6
rp-3d?.....	6
rp-acoustics?.....	8
rp-axi?	6
rp-des?	7
rp-double?.....	6
rp-dual-time?.....	7
rp-electro-chem?	8

rp-inviscid?	7
rp-ke?	7
rp-kklw?	7
rp-kw?	7
rp-lam?	7
rp-les?	7
rp-potential?	8
rp-react?	8
rp-sa?	7
rp-sas?	7
rp-seg?	7
rpsetvar	6, 24
rp-spe?	7
rp-spe-part?	8
rp-spe-site?	8
rp-spe-surf?	8
rp-trans-sst?	7
rp-turb?	7
rp-unsteady?	7
rp-v2f?	7
rp-wallfilm?	8

S

sg-disco?	7
sg-dpm?	8
sg-dtrm?	7
sg-dynmesh?	8
sg-ignite?	8
sg-melt?	8
sg-montecarlo?	7
sg-mphase?	7
sg-node-udm?	8

sg-noniterative?	9
sg-nox?	8
sg-p1?	7
sg-par-premix?	8
sg-pb?	7
sg-pdf-transport?	8
sg-premixed?	8
sg-rosseland?	7
sg-rsm?	7
sg-s2s?	7
sg-solar?	7
sg-soot?	8
sg-sox?	8
sg-spark?	8
sg-swirl?	6
sg-udm?	8
sg-uds?	8
sg-wallfilm?	8
sg-wetsteam?	7
solution/converged?	6
strip-directory	9
surface-id->name	22
surface-name->id	22

T

thread-id->name	19
thread-name->id	19
ti-menu-load-string	4

W

with-output-to-string	4, 12
-----------------------	-------

References

- [1] ANSYS, Inc., "Solution 2042681: Introduction to ANSYS Fluent scripting - Part 1 - Introduction to Scheme," ANSYS, Inc., 2016.
- [2] ANSYS, Inc., "Solution 2042683: Introduction to ANSYS Fluent scripting - Part 3 - Creating keyboard shortcuts, menu items and panels," ANSYS, Inc., 2016.
- [3] ANSYS, Inc., "ANSYS Fluent Customization Manual, Release 17.1," ANSYS, Inc., 2016.
- [4] ANSYS, Inc., "Solution 2004273: How to reset the number of iterations while keeping the solution?," ANSYS, Inc., 2015.
- [5] Ansys, Inc., "Solution 933: How to start V2F turbulence model," ANSYS, Inc., 2015.
- [6] ANSYS, Inc., "Solution 2038952: How to retrieve the case name in UDF?," ANSYS, Inc., 2015.
- [7] ANSYS, Inc., "Solution 2039826: How do I pick a surface or volume integral value to do further calculations in a journal in Fluent?," ANSYS, Inc., 2016.
- [8] ANSYS, Inc., "Solution 2039638: Why does the scheme command (pick-a-real) not work after upgrading to a new Fluent version," ANSYS, Inc., 2016.
- [9] ANSYS, Inc., "Solution 2041732: What is the difference between the interior of a cell zone and the Cell Zone in Fluent?," ANSYS, Inc., 2016.
- [10] ANSYS, Inc., "Solution 2014956: Computing time taken for execution of Fluent commands," ANSYS, Inc., 2016.
- [11] ANSYS, Inc., "Solution 2042751: How to remove all default interior zones from ANSYS Fluent post-processing?," ANSYS, Inc., 2016.
- [12] ANSYS, Inc., "Solution 2039572: How to change multiple ANSYS Fluent boundary conditions that follow a name pattern simultaneously?," ANSYS, Inc., 2016.
- [13] Ansys, Inc., "Solution 2042772: How to create many equally spaced (iso-) surfaces for ANSYS Fluent postprocessing?," ANSYS, Inc., 2016.
- [14] ANSYS, Inc., "Solution 912: How to create an iso-surface along an arbitrarily oriented plane?," ANSYS, Inc., 2015.
- [15] ANSYS, Inc., "Solution 1030: How to automatically create multiple point monitors?," ANSYS, Inc., 2015.
- [16] M. Javurek, „Fluent Scheme Dokumentation,“ Johannes Kepler University Linz, Institute of Fluid Mechanics and Heat Transfer, Linz, 2011.

Keywords: Best-Practice; best practices; guidelines; guideline; Scheme; Script; Scripting; ANSYS Fluent; Journal; Journaling; TUI; Text User Interface; Automation

Contributors:

- Akram Radwan, Senior Technical Support Engineer, ANSYS Germany, Customer Excellence, European Technology Group
- Dr.-Ing. Amine Ben Hadj Ali, Senior Technical Support Engineer, ANSYS Germany, Customer Excellence