ANSYS Fluent CFD—Text User Interface & Scheme for Automation

Scheme Basics

Practice Scheme online at repl.it/languages/Scheme.

```
(define symbol value)     (set! symbol newvalue)
```
Create a new global variable `symbol = value` or assign to it.
```
(define x 3)              (set! x 4)
```
```
(let ((symbol1 value1) (symbol2 value2)) ... )
```
Create a scope with local variables. (x has no value out of scope.)
```
(let ((x 3)) x)
```
```
(+ x y)      (- x y)      (* x y)      (/ x y)      (expt x y)   (< x y)      (= x y)
```
Operate on several values and return the result.
```
(+ 2 4)      (- 5 3)      (* 1.25 6)   (/ 14 3.5)   (expt 5 3)   (< 5 3)      (= 4 4)
```
```
'(x y z)     (cons w '(x y z))                                  (append '(x y z) '(t u v))
```
Creates, expands, or appends a list (single quote forces non-evaluation of parentheses).
```
'(5 6 7)     (cons 4 '(5 6 7))  (cons '(5 6 7) '(4))      (append '(4 5 6) '(7))
```
```
(car '(x y z))     (cdr '(x y z))                          (length '(x y z))
```
Extract first element of list, or return list without first element; get list length.
```
(car '(5 6 7))     (cdr '(5 6 7 8))                        (length '(1 2 3))
```
```
(and x y)    (or x y)     (not x)      (list? x)    (real? x)    (positive? x)
```
Perform Boolean logic or discover value properties (many others exist).
```
(list? '(5 6 7))   (and #t #f)  (or #t #f)
```

Looping

```
(do ((x x_lo (+ x x_delta))) ((> x x_hi)) ... )
```
A `do` loop carries out a repeated task at intervals `x_delta`. Analogous to `for` loop in languages like C.

Display values from –1.0 to 1.0 at a spacing of 0.25:
```
(do ((z -1 (+ z 0.25))) ((> z 1)) (display (format "~a " z) ) )
```
```
(for-each function list1 list2 ...)
```
A `for` loop performs a self-defined `function` for each element of `list1`, `list2`, etc. The number of arguments to `function` must correspond to the number of lists.

Multiply numbers together across two arrays:
```
(define (mult x y) (* x y))  (for-each (mult) '(0 0.25 0.5 0.75 1.0) '(1 2 3 4 5))
```

Control

```
(if test true_value false_value)
```
An `if` statement allows you to execute code conditionally.

If a value is positive, decrement it.
```
(define x 5) (if (not(positive? x)) '() (set! x (- x 1)))
```
```
(cond (test1 true_value)
      (test2 true_value)
      (else  false_value))
```
A `cond` statement replicates the `switch/case` behavior of C, obviating the need for iterated `if`s.

Functions

```
(define symbol value)
```
Create a function returning the absolute value of parameter `x`*:*
```
(define (abs x)
    (cond ((< x 0.0) (- x))
          ((> x 0.0) x)))
```
Create a function returning the sum of a list parameter `ls`*:*
```
(define (sum ls)
    (if (null? ls) 0 (+ (car ls) (sum (cdr ls)))))
```
Based on Mirko Javurek, *Scheme Programing in FLUENT 5 & 6*, 2004 (tr. Tiago Macarios), and Jerry Cain, CS 107 Scheme Handout (Stanford).

Fluent Journals

Fluent can record your actions in the graphical user interface and text user interface as a journal file. GUI-generated journals are verbose and opaque—it is more straightforward to use pure TUI language to parameterize a simulation. The following carry out equivalent actions in loading a mesh:

```
(cx-gui-do cx-activate-item "MenuBar*ReadSubMenu*Case...")
(cx-gui-do cx-activate-item "Select File*Text" "airfoil.msh")
(cx-gui-do cx-activate-item "Select File*OK")
```

```
(file read-case airfoil.msh)
```

Fluent TUI

The Fluent Text User Interface (TUI), supports Fluent Scheme as outlined above and in Javurek's guide. All Fluent variables and functions are available in the environment, which treats the menus as a Unix-like file system. A large number of specialized functions are supported; consult the *Fluent Text Command List* for details. Some common commands include:

On a line by itself, list currently accessible menu commands.

```
(rpgetvar 'var_name)
(rp-var-define 'var_name value)       (rpsetvar 'var_name value)
```

Show or set value of `var_name`, a model-related variable (RP refers to user-specifiable model macros).

```
(rpgetvar 'flow-time)           (rpsetvar 'flow-time 0)
```

```
(cxgetvar 'var_name)
(cx-var-define 'var_name value)       (cxsetvar 'var_name value)
```

Show or set value of `var_name`, an environmental Fluent-related variable (Cortex).

```
(cxgetvar 'CMAP-list)           (cxsetvar 'def CMAP "rgb")
```

```
(rpgetvar 'var_name)            (rpsetvar 'var_name value)
```

Show or set value of `var_name`, a model-related variable.

```
(rpgetvar 'flow-time)           (rpsetvar 'flow-time 0)
```

```
the-environment
```

The TUI provides a standard environment to interpret (`evaluate`) values within.

Evaluate a list as a Scheme command in the TUI:

```
(define y '(1 2 3)) (eval y (the-environment))
```

Test whether a symbol has been defined in the TUI:

```
(symbol-bound 'symbol (the-environment))
```

Test whether a variable has been assigned in the TUI:

```
(symbol-assigned 'symbol (the-environment))
```

```
(format # f "... ~a ~d ~s ..." var_real var_int var_string)
```

Output a formatted string (rather like C `printf`). The format strings are not well documented.

```
(system "command")
```

Run command in the system shell.

```
(system "ls ..")
```

Create a function returning the name of each boundary condition:

```
(define (export-bc-names)
    (for-each (lambda (name) (display (format #f " {~a, \"~a\", \"~a\"},\n" (zone-name->id
name) name (zone-type (get-zone name))))) (inquire-zone-names)))
(export-bc-names)
```

A common requirement of Fluent users is the parameterization of simulations. In order to do this in Fluent, one needs to either write a Scheme loop within a journal file or write a Python/Perl/Ruby script to generate a script for each set of parameters *a*, *b*, *c* and call Fluent on the command line.

```
$ fluent 3d < job-all.jou >& job.log &
```

```
$ fluent 3d < job-a-b-c.jou >& job.log &
```