目录

1. 前言.....3
2. 简介.....3
3. Fluent-Scheme 接口.....3
   - 3.1 在 Fluent 中调用 scheme 命令.....3
   - 3.2 在 Scheme 中调用 Fluent 命令.....3
   - 3.3 RP-变量.....4
   - 3.4 CX-变量.....4
4. Fluent-Scheme-UDFs 接口.....4
   - 4.1 数据交换.....4
   - 4.2 函数的调用.....5
5. 数学函数.....5
6. 全局 Scheme 变量.....6
7. 局部 Scheme 变量.....7
8. 串列(list).....7
9. If 命令.....8
10. Do 循环.....9
11. Format 命令.....11
12. for-each 循环.....11
13. 在 TUI 中的 Alias 方法.....12
14. 实例：建立动画.....13
15. 实例：从数据文件生成报告.....14
16. 实例：从 Data 或者 Case 文件你读取数据.....17
17. 实例：输出 Fluent Zone 的名称到 UDF.....18
18. 迭代控制.....20
19. Fluent Scheme 的特色.....21
    - 19.1 Eval 命令和环境.....21
    - 19.2 Listen 命令.....21
    - 19.3 Format 命令.....22
    - 19.4 System 命令.....22
    - 19.5 Fluent 变量和函数.....22
20. Scheme 文献.....22
21. Fluent Scheme 的标准函数.....22
22. Fluent-Scheme 环境.....26

1. 前言

在 Fluent 中使用 Scheme 可以非常轻松的自动执行仿真流程。非常遗憾的是，到目前为止都没有正式的支持文档。在 Fluent 中使用 Scheme 必须对标准的 Scheme 语言有足够的了解。这个文档非常简单，但是仍然不失为一个好的参考资料，Fluent 德国都是直接向用户推荐这个手册，因为他们自己本身也没有官方的手册。实际上，Fluent 的后续版本将不会再使用 Scheme 作为开发语言，转而使用 Python 这个灵活度更高的语言。

2. 简介

Scheme 是 Lisp 的一个分支，有着非常统一而又简单的命令格式：

(commandname argument1 argument2 ...)

每一个命令调用都是一个函数调用，因此也就会输出一个结果。命令名和变量名不区分大小写，但是只能以字母开头，可以包含除了 a-z 和 0-9 之外的特殊符号，包括+-*/<>=?.:%$!~^_等字符。

注释使用;;开头，行结束就表示注释结束。

3. Fluent-Scheme 接口

3.1 在 Fluent 中调用 scheme 命令

- 使用 Fluent 的命令行界面输入（也可以使用鼠标拷贝命令）或者
- 在文本编辑器中编好 Scheme 程序，然后用.scm 结尾的文件储存，再通过 Fluent 的菜单“ File/Read/Scheme ”调用
- 如果在用户文件夹中存在一个.fluent 文件，这个文件会随着 Fluent 的启动而运行
- 在菜单 ” Solve/Monitor/Commands/Command “ 输入 Scheme 命令，可以在每次迭代或者时长运行。

3.2在 Scheme 中调用 Fluent 命令

- 文字界面输入命令:

(ti-menu-load-string "display/contour temperature 30 100")

返回值： #t 代表执行成功，#f 代表失败或者是使用 Ctrl-C 取消的；Ctrl-C 终止 Fluent 命令，但不终止 Scheme 程序的执行。

- GUI 命令：Journal 文件中可以直接使用 Scheme 命令，执行图形化操作。比如：

(cx-gui-do cx-activate-item "Velocity Vectors*PanelButtons*PushButton1(OK)")

文字命令执行速度更快，更紧凑，而且在很多方面可以使用。GUI 命令比较慢，不方便查看和编写(查找 Fluent 命令比较麻烦)。因此应该优先使用文字界面命令，只有当没有文字

命令可以使用的时候才使用 GUI 命令。
文字界面命令还不能注释，操作步骤：搜索想要运行的命令，尝试运行，然后组织命令行。

在 Fluent 文字界面的输出样式：

```
(display object)
(newline)
```

3.3 RP-变量

获得变量值：比如获得模拟时间:

```
> (rpgetvar 'flow-time)
0.1
```

设置变量值:

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

所有 RP 变量都是在 Case 中定义(参考"变量"一章).

3.4 CX-变量

读取变量，比如说颜色历程表,

```
> (cxgetvar    'cmap-list)
0.1
```

设置变量:

```
> (cxsetvar 'def-cmap "rgb")
```

所有的 CX 变量都是在 Case 文件中定义的（参考 ”Cortex 变量 ”）

4. Fluent-Scheme-UDFs 接口

4.1 数据交换

可以定义自己的 RP 变量，通过 Fluent 的文字界面或者在 UDF 中通过特殊的函数进行调用。
定义一个自己的 RP 变量：:

```
(rp-var-define name default-and-init-value type #f)
```

类型: 'int 'real 'boolean 'string ...?
比如:

```
> (rp-var-define 'udf/var1 0 'real #f)
```

变量信息:

```
> (rp-var-object 'udf/var1)
(udf/var1 0 real #f 0)

> (rp-var-object 'udf/var2)
#f
```

变量的改变和查询可以像上面一样通过 rpsetvar 和 rpgetvar 来完成。

如果一个变量已经被定义一次，则在 Fluent 程序结束之前它都是有效的，储存在每个 Case 文件里面。重新载入这样的一个 Case 文件的时候，如果此变量没有被定义，则它会被自动创建，并且储存在 Case 文件里面的变量值会被指定给这个变量。

在 UDF 里面，RP 变量可以通过如下的 C 函数（在 Fluent.Inc/fluentX.Y/src/var.h 中被声明）赋值和查询

```
real RP_Get_Real(char *s);
long RP_Get_Integer(char *s);
char *RP_Get_String(char *s);
boolean RP_Get_Boolean(char *s);

void RP_Set_Real(char *s, real v);
void RP_Set_Integer(char *s, long v);
void RP_Set_Boolean(char *s, boolean v);
void RP_Set_String(char *s, char *v);
void RP_Set_Symbol(char *s, char *v);
```

例如：

```
var1 = RP_Get_Real("udf/var1");
RP_Set_Real("udf/var1", 3.2);
```

如果 UDF 是在并行模式中被使用，RP 变量的操作需要特别注意，详情请查阅 Fluent 的 UDF 手册。

4.2 函数的调用

EOD 类型的 UDF 可以使用 Scheme 的如下命令调用：

```
(%udf-on-demand "udf-eod-name")
```

目前还没有可能在在 UDF 中调用 Scheme 函数；虽然 Fluent.Inc/fluentX.Y/cortex/src/cx.h 中声明的 C 函数 CX_Interpret_String("scheme-command-string")可以解释"scheme-command-string",但是没有途径可以进入此环境。

5. 数学函数

基本运算 + - * /, 可以有 2 个以上的参数:

```
> (+ 2 4 5)
11

> (/ 6 3)
2

> (/ 2)                ;; 等同于 (/ 1 2)
0.5
```

其它函数： (abs x), (sqrt x), (expt x y) [= xy], (exp x)[= ex], (log x) [= ln x], (sin x), (cos x), (atan x), (atan x y) [= arctan(x/y)], …

取整函数:

```
> (remainder 45 6)
3

> (modulo 5 2)
1

(truncate x), (round x), (ceiling x), (floor x), ...
```

其它

```
(max x y ...), (min x y ...)
```

比如，寻找串列中的最大值:

```
> (apply max '(1 5 8 3 4))
8
```

6. 全局 Scheme 变量

用如下方法定义:

```
> (define x 3)
> (+ x 1)
4
```

没有变量类型区别 (Integer, Real, String, ...) – 每个变量都可以接受任何一种类型的值.
使用再定义方法改变变量值(不能在函数内部操作，函数内部是局部变量区域，所以定义的将会是一个新的局部变量)，更好的方式是使用如下的句子：

```
(set! x 1)
```

输出显示变量值

```
(display x)
```

或者

(write x)

Write 仅当 Fluent 变量值都储存在文件里面而又想读出来的时候才使用；Write 使用前置符号显示符号变量值。
常量: 整数 (2), 浮点数 (2.5), 布尔值 (#t 真, #f 假) 字符串 ("this is a text string") 和符号（ 'symbol）,比如:

(define x 'this-is-a-symbol)

字符串中的特殊符号使用:

\" "
\n 新行

全局变量和自定义 Scheme 函数的有效性保持到 Fluent 结束运行。

7. 局部 Scheme 变量

```
(let ((var1 value1) (var2 value2) ...)
    ... 命令...
)
```

8. 串列(list)

定义方法，例如：:

> (define my-surfaces '(wall-top wall-bottom symmetry))

任意长度，支持动态管理和嵌套
使用如下格式进行定义： ‘ (elements ...) :

> (define l '(a b c))

串列的第一个元素

```
> (car l)
a
```

串列剩下的部分 (没有第一个元素的串列)

```
> (cdr l)
(b c)
```

串列长度（元素个数）

```
> (length l)
3
```

串列的第 i 个元素

```
(listref liste i)
```

在串列中搜索元素:

```
> (member 'd '(a b c d e f g))
(d e f g)
```

对串列应用函数：

```
> (map (lambda (x) (* x x)) '(1 2 3))
(1 4 9)
> (apply + '(2 4 5))
11
> (apply max '(1 5 8 3 4))
8
```

9. If 命令

If 命令是一个函数：

```
(if cond true-value false-value)
```

Cond 是一个逻辑表达式，#t (true) 或者 #f (false).
比较操作：
等式:

```
(= a b)   ;; 数字
(eq? a b)   ;; 对象
(eqv? a b) ;; 对象比较值
```

关系：

```
(positive? x)
(negative? x)
(< a b)
(> a b)
(<= a b)
(>= a b)
```

逻辑函数:

```
(not a)
(and a b c ...)
(or   a b c ...)
```

if 和 else 分支可以实用块命令 begin 进行很多扩展

```
(if cond
    (begin   ;; if
```

```
        ...
        true-value
    )
    (begin    ;; else
        ...
        false-value
    )
)
```

如果 if 命令的返回值不需要保存，else 分支和它的返回值都可以省略。

用于条件流程控制的复合命令（例如分段函数）

(cond (test1 value1) (test2 value2) ... (else value))

一个变量的离散值

(case x ((x11 x12 x13 ...) value1) ((x21 x22 x23 ...) value2) ... (else value))

如果 X 可以在串列中找到（例如在 (x11 x12 x13 ...)），则对应的值赋给(value1)。

10.Do 循环

最简单的形式（变量，开始值，指定步长，结束条件）

(do ((x x-start (+ x delta-x))) ((> x x-end)) ...loop-body...)

例子：创建 ISO 面，等间距创建多个 ISO 面并自动命名。首先必须在 TUI(Text User Interface)中形成一个用于创建 ISO 面的脚本集。

```
>
adapt/                    grid/                    surface/
display/                  plot/                    view/
define/                   report/                  exit
file/                     solve/

> surface

/surface>
delete-surface            mouse-line               point-array
surface-cells             mouse-plane              rake-surface
iso-surface               mouse-rake               rename-surface
iso-clip                  partition-surface        sphere-slice
list-surfaces             plane-slice              zone-surface

/surface> iso-surface

iso-surface of>
pressure                  entropy                  x-surface-area
pressure-coefficient      total-energy             y-surface-area
dynamic-pressure          internal-energy          z-surface-area
```

```
...
rel-total-temperature      x-coordinate                dp-dx
wall-temp-out-surf          y-coordinate                 dp-dy
wall-temp-in-surf           z-coordinate                 dp-dz

iso-surface of> x-coordinate
new surface id/name [x-coordinate-31] testname
range [-10.0131, 4.8575001]
from surface [()] ()
()
iso-value(1) (m) [()] 1.234
iso-value(2) (m) [()] ()
```

所有命令写成一行（用逗号取代回车）如下：

```
surface/iso-surface x-coordinate testname () 1.234 ()
```

进行参数循环：

```
(do ((x 0 (+ x 0.2)) ) ((> x 3.1))
   (ti-menu-load-string
      (format #f "surface/iso-surface x-coordinate x-~3.1f () ~a ()" x x))
)
```

生成如下的命令行

```
surface/iso-surface x-coordinate x-0.0 () 0 ()
surface/iso-surface x-coordinate x-0.2 () 0.2 ()
surface/iso-surface x-coordinate x-0.4 () 0.4 ()
...
surface/iso-surface x-coordinate x-3.0 () 3 ()
```

细化：对正负坐标进行更好的命名

```
  (do ((z -1 (+ z 0.25))) ((> z 1))
      (ti-menu-load-string
         (format #f "surface/iso-surface z-coordinate z~a~05.3f () ~a ()"
             (if (>= z 0) "+" "") z z))
)

surface/iso-surface z-coordinate z-1.000 () -1 ()


surface/iso-surface z-coordinate z-0.750 () -0.75 ()
surface/iso-surface z-coordinate z-0.500 () -0.5 ()
surface/iso-surface z-coordinate z-0.250 () -0.25 ()
surface/iso-surface z-coordinate z+0.000 () 0 ()
surface/iso-surface z-coordinate z+0.250 () 0.25 ()
surface/iso-surface z-coordinate z+0.500 () 0.5 ()
surface/iso-surface z-coordinate z+0.750 () 0.75 ()
surface/iso-surface z-coordinate z+1.000 () 1 ()
```

变化：两个循环变量

```
(do ((x 0 (+ x 0.2)) (i 1 (+ i 1))) ((> x 3.1))
   (ti-menu-load-string
      (format #f "surface/iso-surface x-coordinate x-~02d () ~a ()" i x))
)

surface/iso-surface x-coordinate x-01 () 0 ()
surface/iso-surface x-coordinate x-02 () 0.2 ()
surface/iso-surface x-coordinate x-03 () 0.4 ()
...
surface/iso-surface x-coordinate x-16 () 3 ()
```

11.Format 命令

(format #f "像 C 语言一样的格式化字符串" var1 var2 ...)

和 C 不一样的是，不使用%符号开头，而是使用~符号开头；比如说：

~a
~d

通用形式的任意变量（字符串没有“ “ ）

整数
~04d 整数，占 4 个位置，不足的以 0 在前面填充。如输出为 0005， 对于文件名命名很重要。

~f 浮点小数
~4.2f 浮点数，总共占 4 个位置，2 位小数。如 1.2 wird zu 1.20
~s

带双引号的字符串。如(format #f "string: ~s !" "text") 输出字符串 "text" !
...

特殊符号：
\n 回车符
\" "

format 命令和它的模式不是标准的 Scheme 语法，是依赖于 Fluent 的 Scheme 解释器。

12.for-each 循环

针对一个 List 的一个或者多个元素都运行一个自定义的函数：

(for-each function list1 list2 ...)

函数的参数个数必须和 List 的个数对应。设置多个墙区的边界条件时，需要设置 Zonename 或者 ZoneID、文件名字（不含大写字母）、温度以及墙的速度等等。

```
(define velocity 0.1)
(for-each
   (lambda (zone)
      (ti-menu-load-string
         (format #f "def/bc/wall ~a 0 0 yes giesspulver yes temperature no 1800 yes no no ~a
0 -1
0    no 0 0.5" zone velocity)
      )
      (newline) (display "    ")
   )
   '(
      kok_li kok_re
      kok_innen kok_aussen
      bieg_li bieg_re
      bieg_aussen bieg_innen
      kreis_li          kreis_re
      kreis_aussen    kreis_innen
   )
)
```

Lambda 命令用来定义局部函数

```
(lambda (arg1 arg2 ...)    ... Funktionswert)
```

13.在 TUI 中的 Alias 方法

在 TUI 中创建缩写命令：

```
(alias 'name scheme-function)
```

例如：

```
(alias 'time (lambda () (display (rpgetvar 'flow-time))))
```

在命令行界面中调用：

```
> time
  0.1
```

参数不能在 Scheme 函数中直接给出(总是空参数，Lambda())，而是必须通过在命令行中使用如下函数进行读取：

```
(read-real prompt default)
(read-integer prompt default)
(ti-read-unquoted-string prompt default)
(yes-or-no? prompt default)
```

prompt 是一个字符串，default 代表缺省值，代表如果用户只是按回车的时候的返回值。

如果你在.fluent-file 中定义了各种缩写替代命令，那么就可以直接使用。

14.实例：建立动画

从一个静态计算的数据文件中提取的图片组成一个动画。数据文件名称采用数字进行编号，开头、结尾是固定的，中间部分逐步增长。 运行 Fluent 命令的时候碰到的错误，或者按下 Ctrl-C 可以终止此 Scheme 程序。

```
(define datfilename "test-")      ;; -> test-0010.dat, test-020.dat, ...
(define first-index    10)
(define last-index    110)
(define delta              10)
(define imagefilename "image-")      ;; -> image-01.bmp, ...

(define (time) (rpgetvar 'flow-time))
(define t0 0)


;;-------------------------------------------------------------------------
;; funktion, die die einzelbilder fuer den film erstellt
;;-------------------------------------------------------------------------
(define (pp)
   (let
      (
         (break #f)
      )
      (ti-menu-load-string "display/set/hardcopy/driver/tiff") ;; TIFF-Format einstellen
      (ti-menu-load-string "display/set/hardcopy/color-mode/color") ;; Default ist "grey"
      (do      ((j first-index (+ j delta))      ;; datfile      startwert und delta
                  (i    1 (+ i    1)))                    ;; imagefile startwert und delta
               ((or (> j last-index) break))    ;; datfile      endwert
         (set! break (not (and
            (ti-menu-load-string
               (format #f "file/read-data ~a~04d.dat" datfilename j))
            (begin (if (= i 1) (set! t0 (time))) #t)
            (disp)
            (system "rm temp.tif") ;; hardcopy funktioniert nicht wenn file schon existiert
            (ti-menu-load-string "display/hardcopy temp.tif")
            (system
                  (format #f "convert temp.tif ~a~02d.bmp &" imagefilename i))
                  ;; convert-Befehl von www.imagemagick.com
         )))
      )
      (if break (begin (newline)(newline)(display "scheme interrupted!")(newline)))
   )
)
```

示例函数(disp)：简单的云图显示

```
(define (disp)
   (ti-menu-load-string "display/contour/temperature 290 1673")
```

)
示例函数(disp)：叠加云图/速度矢量图，插入时间：

```
(define (disp)
   (and
      (ti-menu-load-string
         (format #f "display set title \"Time = ~5.1f s\"" (- (time) t0))
      (ti-menu-load-string "display/set/overlays no")
      (ti-menu-load-string "display/contour temperature 290 1673")
      (ti-menu-load-string "display/set/overlays yes")
      (ti-menu-load-string "display/velocity-vectors velocity-magnitude 0.0 1.0 5 0")
                                                                  ;; colored by              min max scale
skip
   )
)
```

示例函数(disp)：生成等值面， VOF 计算产生的相边界使用 y 坐标(高度)进行颜色标注：

```
(define (disp)
   (and
      (ti-menu-load-string "display/surface/iso-surface vof-steel interface-1 , 0.5 ,")
      (ti-menu-load-string "display/set/contours/surfaces interface-1 ()")
      (ti-menu-load-string "display/contour y-coordinate 2.755 2.780")

      (ti-menu-load-string "display/surface/delete interface-1")
   )
)
```

调用 disp 函数测试：

```
> (disp)
```

调用函数创建图片

```
> (pp)
```

15.实例：从数据文件生成报告

必须通过一个 transcript 文件来产生数据报告：

```
   (ti-menu-load-string "file/start-transcript temp.trn")
   (ti-menu-load-string "report/cell-average fluid , temperature")
   (ti-menu-load-string "file/stop-transcript")
```

对此 Scheme 有一个函数可以起到同样的作用：

```
   (with-output-to-file "temp.trn"
      (lambda ()
         (ti-menu-load-string "report/cell-average fluid , temperature")))
```

这里运行成功不会在屏幕输出任何东西。

Transcript 文件 "temp.trn":

```
report/cell-average fluid , temperature
  volume-average of temperature on cell zones (fluid)
  Volume-weighted average = 300
file/stop-transcript
```

在 Scheme 中 Transcript 文件作为一个 List 的对象读入，搜索“ = “ 后面的内容，输出跟在等号后面的元素的对应值：

```
  (let
```

```
    (
      (data
        (let ((p (open-input-file "temp.trn")))
          (let f ((x (read p)))
            (if (eof-object? x)
              (begin
                (close-input-port p)
                '())
              (cons x (f (read p)))
            )
          )
        )
      )
      (value 0)
    )
    (ti-menu-load-string "! rm temp.trn") (newline)

    (do ((i 0 (+ i 1)) ) ((>= i (length data)))
      (if (eq? (list-ref data i) '=)
        (set! value (list-ref data (+ i 1)))
      )
    )
    value
  )
```

缩写 do 循环（没有变量需要取值）的方法：

```
    (cadr (member '= data))
```

嵌套 car-cdr：

```
(cadr x) = (car (cdr x))
```

输出基于热流平衡（在 TUI 中为所有面建立）的特定表面的热流值：

输出的格式：

```
...
zone 15 (stahl-bodenplatte): 11.2
zone 5 (stahl-kokille): 53.5
zone 6 (schlacke-aussen): 32.4
zone 14 (haube-schlacke): 26.9
...
```

Scheme 程序：

```
(let
  (
    (p (open-output-file "fluxes.txt")) ;; Ausgabe-Textdatei öffnen
    (n 0)
    (surfaces '(
      stahl-bodenplatte
      stahl-kokille
      stahl-schlacke
      haube-schlacke
      elektrode-schlacke
```

```
        schlacke-innen
        schlacke-aussen
        aufsatz-schlacke
        haube-kuehlung
    ))

)
(for-each
    (lambda (filename)
        (if (zero? (modulo n 2))    ;; nur jedes zweite Datenfile nehmen
            (begin
                (ti-menu-load-string
                    (format #f "file read-data ~a" filename))

                (ti-menu-load-string "file/start-transcript temp.trn")
                (ti-menu-load-string "report/heat-transfer")
                (ti-menu-load-string "file/stop-transcript")
                (define data                                    ;; transcriptfile in "data" laden
                    (let ((p (open-input-file "temp.trn")))
                        (let f ((x (read p)))
                            (if (eof-object? x)
                                (begin
                                    (close-input-port p)
                                    '())
                                (cons x (f (read p)))
                            )
                        )
                    )
                )
                (ti-menu-load-string "! rm temp.trn")

                (display (time) p)
                (display "      " p)
                (for-each
                    (lambda (zone)
                        (begin
                            (display (list-ref (member (list zone) data) 2) p)
                              ;; fluxwert von zone ermitteln
                            (display " " p)
                        )
                    )
                    surfaces
                )
                (newline p)
            )
        )
        (set! n (+ n 1))
    )
    '(
        best-0060.dat    best-0120.dat    best-0132.dat    best-0144.dat    best-0156.dat
        best-0168.dat    best-0180.dat    best-0192.dat    best-0204.dat    best-0216.dat
        best-0228.dat    best-0240.dat    best-0252.dat    best-0264.dat    best-0276.dat
        best-0288.dat    best-0300.dat    best-0312.dat    best-0324.dat    best-0336.dat
```

```
)
  )
  (close-output-port p)
)
```

数据文件 list 可以在 Unix 使用 ls –x *.dat 命令建立

16.实例：从 Data 或者 Case 文件你读取数据

Fluent 文件的格式就是嵌套的 Scheme List，比如说：

```
(0 "fluent5.3.18")

(0 "Machine Config:")
(4 (23 1 0 1 2 4 4 4 8 4 4))

(0 "Grid size:")
(33 (10540 21489 10947))



(0 "Variables:")
(37 (
        (flow-time 3.7)
        (time-step 0.1)
        (periodic/pressure-derivative 0)
        (number-of-samples 0)
        (dpm/summary ())))

(0 "Data:")
(2300 (1 1 1 0 0 1 430)
...
```

因此可以非常简单的作为 Scheme 对象读入。 比如说，从数据文件里面读取时间，然后使用 hh:mm:ss 的时间格式重命名数据文件。

```
(let ((p (open-input-file filename)) (found #f) (t -1))
    (do ((x (read p) (read p))) ((or found (eof-object? x)) (close-input-port p) )
        (if (eqv? (car x) 37) ;; variables
            (begin
                (for-each
                    (lambda (y)
                        (if (eqv? (car y) 'flow-time)
                            (begin
                                (set! found #t)
                                (set! t (cadr y))
                            )
                        )
                    )
                    (cadr x))
                (newline)
            )
        )
    )
```

```
    (ti-menu-load-string (format #f "!mv ~a ~a~a.dat"    filename newdatname (sec->hms t)))
)
```
函数 sec->hms 把秒数转换成为 hh:mm::ss 的形式：
```
(define (sec->hms t)
  (let*
    (
      (h (truncate (/ t 3600)))   (t1 (- t   (* h 3600)))
      (m (truncate (/ t1 60)))
      (s (truncate (- t1 (* m 60))))
    )
    (format #f "~02d:~02d:~02d" h m s )
  )
)
```

17.实例：输出 Fluent Zone 的名称到 UDF

UDF 中可以使用 THREAD_ID(t) und THREAD_TYPE(t)，也就是一个边界区的 ID 和类型，而不是直接引用她的名称。

使用下面的 Scheme 函数创建一个数据结构，然后就可以在 UDF 中复制这个数据结构。

```
(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)
  )
)
```

在 Fluent 中运行：
```
(export-bc-names)
  {26, "wall-wehr-l-shadow", "wall"},
  {2, "fluid", "fluid"},
  {29, "wall-damm-l-shadow", "wall"},
  {15, "wall-damm-l", "wall"},
  {17, "inlet", "mass-flow-inlet"},
  {25, "default-interior", "interior"}
  ...
```
这些文字必须复制到下面的 UDF 代码中：
```
#define nc 100
typedef struct zone_info_struct
```

```
{
   int id;
   char name[nc];
   char type[nc];
}
zone_info;

zone_info zone[]={
/*** ab hier aus Fluent-Textinterface kopiert ***/
   {26, "wall-wehr-l-shadow", "wall"},
   {2, "fluid", "fluid"},
   {29, "wall-damm-l-shadow", "wall"},
   {15, "wall-damm-l", "wall"},
   {17, "inlet", "mass-flow-inlet"},
   {25, "default-interior", "interior"}
   ...
};
#define n_zones (sizeof(zone)/sizeof(zone_info))
```

现在可以在 UDF 代码中使用 zone[i].name 的形式调用 Zone name.

另外一个替代方法是使用 Scheme 函数，Zone 的名称作为一个字符串储存在 RP 变量里面。这样做的好处是，这个 RP 变量将会一同写入 Case 文件，那么多个不同的 Case 文件使用自己的 UDF 的时候就没有必要重新编译。生成 Case 文件的时候也只需要调用这个 Scheme 函数一次。缺点是：在并行计算的时候 RP 变量使用起来不太方便。

```
(define (bc-names->rpvar)
   (let ((zone-data ""))
      (for-each
         (lambda (name)
            (set! zone-data
               (format #f "~a ~a ~a ~a    " zone-data
                  (zone-name->id name)
                  name
                  (zone-type (get-zone name))
         )))
         (inquire-zone-names)
      )
      (display zone-data)
      (rpsetvar* 'zone-names 'string zone-data)
   )
)
```

Dabei wird folgende Funktion verwendet:

```
(define (rpsetvar* var type value) ;; create cortex variable if undefined
   (if (not (rp-var-object var))
      (rp-var-define var value type #f)
      (rpsetvar var value)
   )
)
```

UDF-代码：

```
#define max_n_zones 200
#define max_zonenamechars 200

/* globale Variablen */
```

```
char zone_name[max_n_zones][max_zonenamechars];
char zone_type[max_n_zones][max_zonenamechars];
#define THREAD_NAME(t) zone_name[THREAD_ID(t)]

/* lokale Variablen */
char *rps,s[1000];
int i,n;

for(i=0; i<max_n_zones; i++) zone_name[i][0]=0; /* initialisieren */
rps = RP_Get_String("zone-names");
while(rps[0])
{
   sscanf(rps,"%s%n", s,&n); rps += n;
   i = atoi(s);
   sscanf(rps, "%s%s %n", zone_name[i], zone_type[i],&n); rps += n;
}
```

这里把宏 Thread_name(t)中直接赋值成为 Zone 名称。

18.迭代控制

只对非静态计算适用；
Cortex 变量也可以重新赋值！
时间 (t):

flow-time

当前时间步长编号： (N):

time-step

步长值 (Δt):

physical-time-step

保存的迭代 List(首先是当前的迭代，然后是之前的)

```
(residual-history "iteration")
```

取出当前的迭代序号：

```
(car (residual-history "iteration"))
```

误差 List(和每一次迭代对应)

```
(residual-history "continuity")
(residual-history "x-velocity")
(residual-history "temperature")
...
```

控制迭代的 TUI 命令（静态或者当前时间步内的非静态计算）：

solve/iterate number-of-iterations

非静态：

solve/dual-time-iterate number-of-timesteps max-iterations

步长必须使用下面的命令设置：

(rpsetvar 'physical-time-step 0.1)

使用这些命令和变量可以进行复杂的迭代程序控制，例如：
非静态：不使用时间步长数，而是使用计算时间间隔
计算中断后，自动继续非常静态计算
根据表格变换步长宽度
自动保存某时间点的计算，文件名可以插入时间代码
自适应步长调整
对那些耗时很长或者很容易中断的计算进行安全备份；保存当前步长的计算结果同时删除上一次的保存文件。

19.Fluent Scheme 的特色

19.1 Eval 命令和环境

(eval expression environemnt)
标准的 Scheme 环境(the-environment)包含所有的针对 Scheme 标准符号的补充符号，也就是说包含所有用户或者 Fluent 定义的变量和函数。当表达式不需要计算值的时候，可以作为一个符号而不是一个空 list 或者#f 保存到环境中。例如：

(define x 3)
(define y '(+ x 2)) ;; y 是一个 list, 包括 +, x, 2 等 3 个元素
(eval y (the-environment)) ;; y 作为一个 List 保存到环境中;结果为 5

可以使用下面的函数来检测一个符号是否已经定义或者是否已经赋值：

(symbol-bound? 'symbol (the-environment))
(symbol-assigned? 'symbol (the-environment))

19.2 Listen 命令

(list-head list n)
(list-tail list n)

19.3 Format 命令

参考前面的章节

19.4 System 命令

Shell 命令，如：
(system "rm temp.jou")

19.5 Fluent 变量和函数

.所有 Fluent 变量和函数都已经在环境中(the-environment)定义，如果没有相关文档，请查看“ Fluent-Scheme 环境“ 这一节。

20.Scheme 文献

非常遗憾没有专门针对 Fluent Scheme 的文献。由于 Scheme 是一门非常强大的语言，可以实现非常多的高难度的诸如人工智能方面的应用，许多有关 Scheme 的文献讲述的东西非常深，同样也适用于 Fluent。Fluent 推荐下面的链接：

- http://www.swiss.ai.mit.edu/projects/scheme
- http://www.schemers.org/

Fluent 的用户中心有许多实例可以参考：

- http://www.fluentusers.com/

21.Fluent Scheme 的标准函数

下面的这些函数是标准的 Fluent Scheme 函数，没有出现在环境中。这个列表是从 Fluent 的程序文件里面抽取出来的，可能不完整。

<
<=
=
>
>=
-
/
*
+
abs
access
acos
and
append
append!
apply
asin

assq
assv
atan
atan2
begin
bit-set?
boolean?
call/ccinput-port?
car
cdr
ceiling
char<?
char=?
char>?
char?
char-alphabetic?
char-downcase
char->integer
char-lower-case?
char-numeric?
char-ready?
char-upcase
char-upper-case?
char-whitespace?
chdir
clear-bit
close-input-port
close-output-port
closure?
closure-body
cond
cons
continuation?
copy-list
cos
cpu-time
debug-off
debug-on
define
display
do
dump
echo-ports
env-lookup
eof-object?
eq?
equal?
eqv?
error
error-object?
err-protect
err-protect-mt
eval

exit
exp
expand-filename
expt
fasl-read
file-directory?
file-exists?
file-modification-time
file-owner
float
floor
flush-output-port
for-each
foreign?
foreign-data
foreign-id
format
format-time
gc
gc-status
general-car-cdr
getenv
hash-stats
if
int
integer?
integer->char
interrupted?
lambda
length
let
let*
list
list-head
list->string
list-tail
list->vector
local-time
log
log10
logical-and
logical-left-shift
logical-not
logical-or
logical-right-shift
logical-xor
machine-id
make-foreign
make-string
make-vector
map
max
member
memq

memv
min
mod
newline
not
nt?
null?
number?
number->string
oblist
open-file
open-input-file
open-input-string
open-output-file
open-output-string
or

output-port?
pair?
peek-char
port-echoing?
port-name
procedure?
procedure-name
putenv
quotient
read
read-char
real?
remainder
remove-file
rename-file
reverse
set!
set-bit
set-car!
set-cc
set-cdr!
set-echo-ports!
sin
sqrt
stack-object
stack-size
string<?
string=?
string>?
string?
string-append
string-ci<?
string-ci=?
string-ci>?
string-length
string->list
string->number

string-ref
string-set!
string->symbol
substring
substring-fill!
substring->list
subvector-fill!
subvector->list
symbol?
symbol-assigned?
symbol-bound?
symbol->string
system
system
tan
the-environment
time
toggle-bit
trace-ignore
trace-off
trace-on
truncate
unix?
valid-continuation
vector?
vector-length
vector-ref
vector-set!
vms?
write
write-char
write-string

22.Fluent-Scheme 环境

下面列出了可以用在 Fluent-Scheme 环境下的函数。这些函数必须带括号使用，而且没有带参数，因为有时候参数可能很长。

solver-command-name fluent
(client-file-version)
(gui-get-selected-thread-ids)
(gui-show-partitions)
(gui-memory-usage)
(grid-show)
(rampant-menubar)
(gui-reload)
(ti-avg-xy-plot)
(ti-2d-contour)
(ti-avg-contour)

(gui-models-radiation)
(set-radiation-model)
(gui-models-species)
(gui-models-multiphase)
(new-phase-name)
(multiphase-model-changed)
(gui-periodic-settings)
(gui-models-solidification)
(gui-models-energy)
(gui-operating-conditions)
(gui-models-solver)
(ti-set-turbo-topo)
(gui-turbo-twod-contours)
(gui-turbo-avg-contours)
(gui-turbo-xyplots)
(gui-turbo-report)
(gui-set-topology)
(ti-turbo-define)
(ti-write-turbo-report)

cxsweep.provided
(cx-delete-keyframe)
(cx-insert-keyframe)
(cx-display-frame)
keyframes list
(create-mpeg)
(mpeg-open)
(play-mpeg)

#t
(ti-compute-turbo-report)
(write-turbo-data)
(gui-turbo-define)
(correct-turbo-defenition)
(delete-turbo-topology)
(define-turbo-topology)
(setturbovar)
(getturbovar)
(add-turbo-post-menu)
(solve-controls-summary)
(gui-solve-iterate)
(gui-solve-controls-mg)
(gui-solve-controls-solution)
(order/scheme-name->type)
(order/scheme-type->name)
order/schemes list

(cx-set-mpeg-compression)
mpeg-options
mpeg-qscale 8
mpeg-bsearch CROSS2

mpeg-psearch EXHAUSTIVE
mpeg-range 8
mpeg-pattern IBBPBB
mpeg-compression? #f
mpeg-command mpeg_encode
(cx-animate)
(cx-gui-animate)
(video-picture-summary)
(video-summary)
(cx-video-use-preset)
(cx-video-show-picture)
(cx-video-set-picture)
(name-list)

cx-video

  n/a
(print-name->attribute-list)
(print-name->pick-name)
(print-name)
(get-eqn-units-patch)
(get-eqn-units-default)
(get-eqn-var-default)

(cx-video-show-options)
(cx-video-set-options)
(cx-video-close)
(cx-video-open)
(cx-video-panel)
(cx-video-enable)
(get-eqn-var)

cx-video

#t
(set-eqn-var)
(get-eqn-index)
(symbol->rpvar)
(inquire-equations)
(gui-solve-set-limits)
(gui-solve-set-ms)
(gui-patch)
(gui-init-flow)
(gui-solar-calculator)
(gui-particle-summary)
(models-summary)
(gui-dpm-sort)
(gui-models-dpm)
(gui-models-viscous)
(gui-user-memory)
(gui-udf-on-demand)
(gui-udf-hooks)
(gui-uds)