ITASCA™

Theory and background 3: Scripting

2019-01-09 – 2019-01-10

Why have a scripting language?

- Automation of workflow is the future
- Can use scripting languages to assign/read almost anything from the models at precise moments in the calculation scheme
- For example,
  - Assign properties from a distribution
  - Track convergence
  - Automatic running and saving of different model cases
  - Automatic creation of plots for all excavation stages

Scripting languages available in FLAC3D

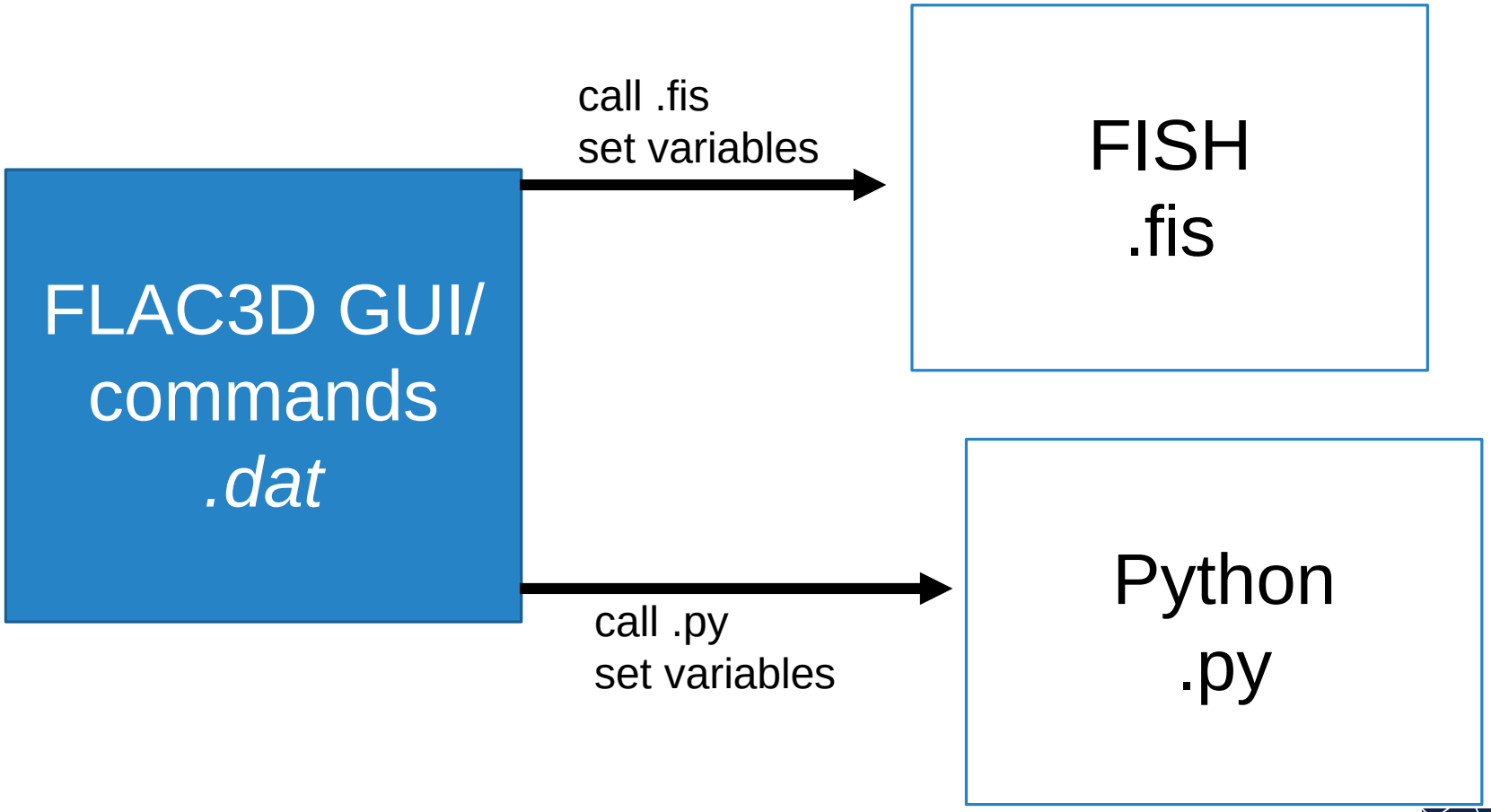
FISH

Python

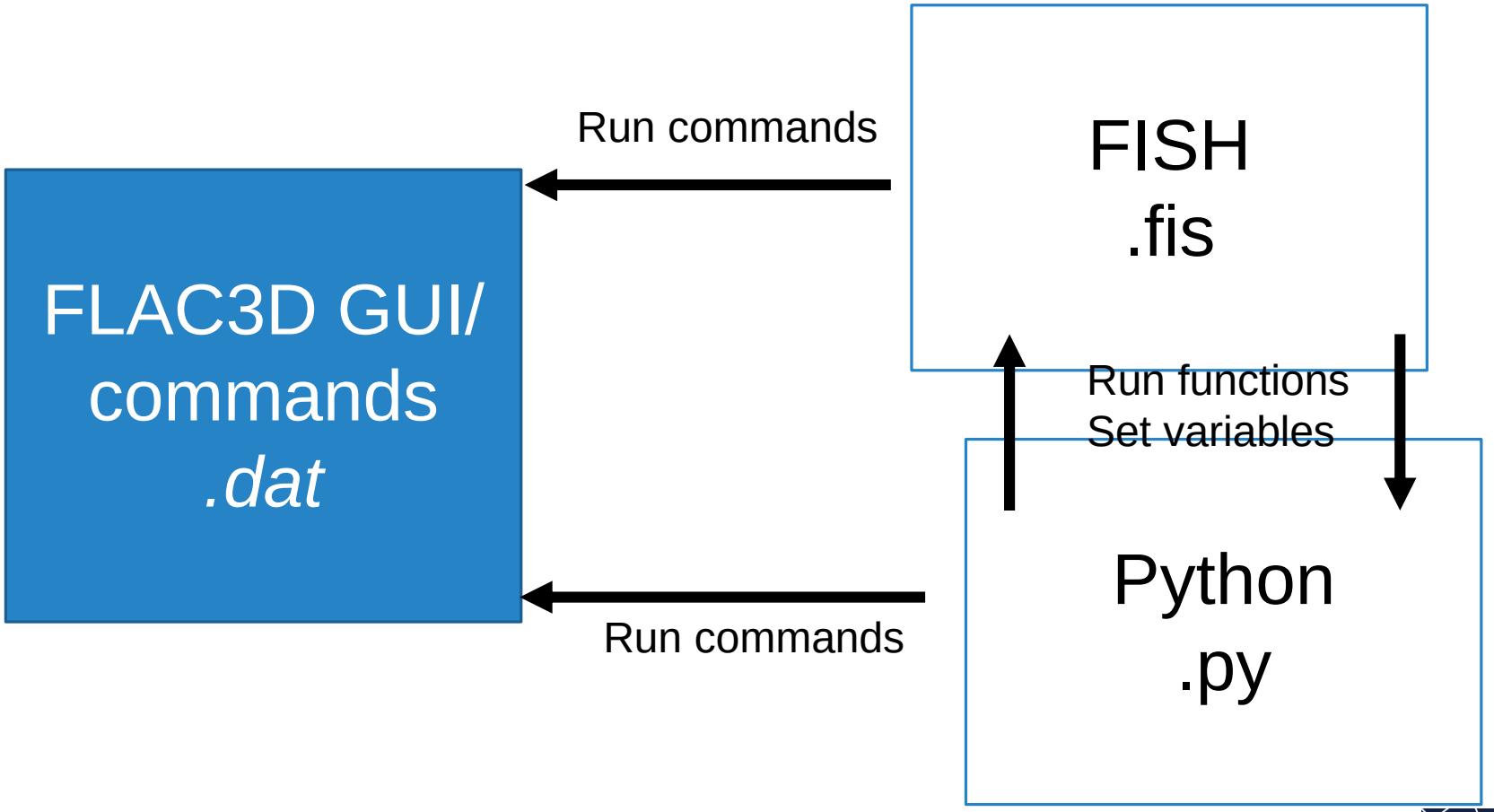
Built in compatibility with model information

Ex, read/write stresses, properties, deformations, etc.

How does everything fit together?



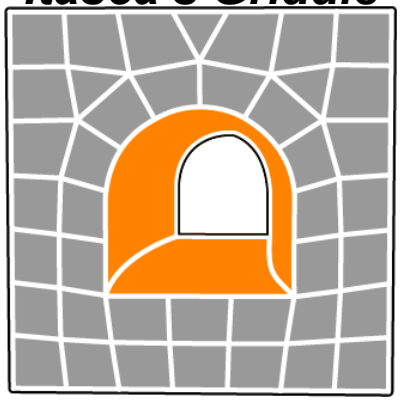
How does everything fit together?



Typical advanced usage

Build geometry → Assign properties, initial stresses, boundary conditions, etc. → Run models (equilibrium, excavation stages), save files, iterate through cases → Post processing

Itasca's Griddle



- Commands
- FISH
- Python

A little extra on FISH…

Introduction

- ***FISH*** is the embedded scripting language for all Itasca programs
  - Stands for “FLACish”, the language of ***FLAC***, the first code it was developed for.
  - Allows access to virtually all internal data structures.

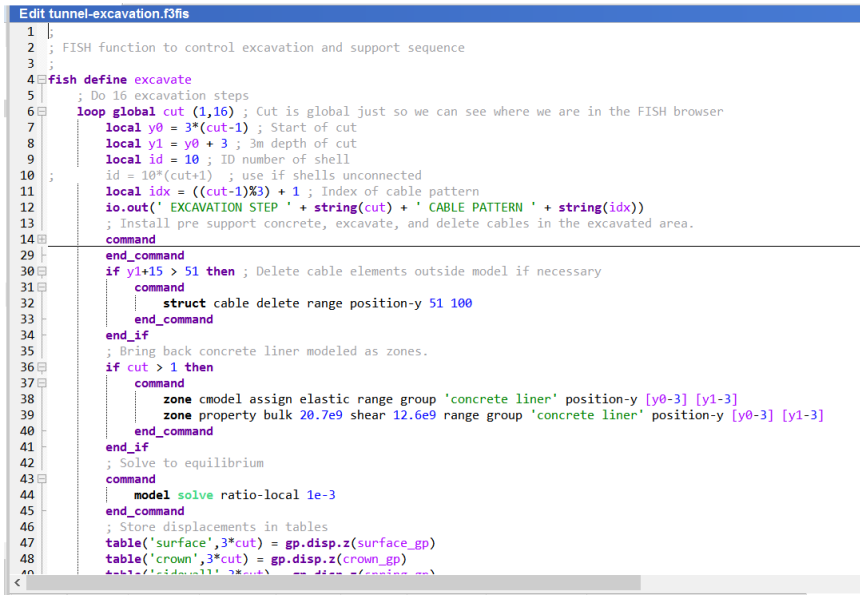
- ***FISH*** is pseudo-compiled into an intermediate language, like Java or .NET
  - We call it “Pseudo-Code” or Pcode. This can be listed and examined.
  - Each function is compiled and stored in the model state.
  - All global symbols are stored in the model state.
  - Variable names and values can be monitored and changed at any time.
  - Help system has Tutorial and Reference

Common ways ***FISH*** is used

- Custom Visualization
- Model Creation
  - Flexible model setup and initialization.
  - Initial stresses, custom geometry, adaptive excavation sequencing
- Model Parameterizaton
  - Setup a model once – run many variations with little to no further effort.
- Multiple Model Run Control
  - Optimizations and Inverse problems
- Physics Extension
  - Add whole new physics to the model – ground freezing, etc.

Writing ***FISH***

- Just like a data file – any text editor can be used.
- Built in text editor offers syntax highlighting, context-sensitive help, code folding, and auto formatting.
- FISH is case insensitive.
- Semi-colon is used for comments – everything after is ignored.
- Command processor recognizes FISH
  - @ prefix
  - [a=b] inline fragments



FLAC3D Editor – example.f3dat

```
1  fish define setup(name)
2      global points = map
3      global solution = (2.0 + math.pi)
4      global cohesion = zone.prop(zone.head,'cohesion')
5      ; Find all gridpoints on footing and store them in
6      global n = 0
7      loop foreach local gp gp.list
8          if gp.isgroup(gp,name) then
9              points(map.size(points)) = gp ; Store
10             local height = gp.pdos.z(gp) ; Store height of footing
11         end_if
12     end_loop
13     ; Find first gridpoint to the right of the footing, at the same height
14     local xnext = 20.0
15     loop foreach gp gp.list
16         if gp.isgroup(gp,name)==false then
17             if gp.pos.z(gp)==height then
18                 xnext = math.min(xnext,gp.p
19             end_if
20         end_if
21     end_loop
22     global width = (3.0 + xnext) * 0.5 ; Adjusted width
23 end
24
```

Strings in green

Keywords and functions in **bold purple**

Bad functions inverted

Comments in gray

Variables in pink

Numbers in blue

FISH Functions -- basic1.f3dat

Edit basic1.f3dat

```
1  ; Basic FISH function definition, enters FISH mode
2  fish define fred ; Func is a global symbol that is ALSO a function
3      a = 3.0      ; 4 space indent is custom, a is
4      fred = a     ; This is how you set the return value
5      ; a = func   ; This would give an error, for recursion
6  end              ; This is how you end the function
7  @fred            ; This calls the function in command mode
8
9  fish define george(a,b) ; Functions can have any number of arguments
10     george = a * b      ; Note a and b are local, and do not appear at right.
11 end
12 list @george(4,5)
13
14 [george(4,5)]
15 [4*5]                   ; Inline FISH is handy!
16
17 ; By default new symbols are globals of integer value 0
18 fish automatic-create off  ; Turns off, all variables must be declared
19                           ; Highly reccomended for large programs
20 fish define mary(c,d)
21     local e = c * d
22     global f = math.sin(e)
23     mart = f  ; Error
24 end
```

FISH Data Types

- Integer: signed 64 bit
- Real: double precision floating point
- String: unicode compliant, can be any length
- Boolean: true or false
- Pointer: refers to an object (piece of data) in the code
- Vector: 2 or 3 dimensional vector
- Tensor: Symmetric 3D tensor (six values).
- Array: one or multi-dimensional array of ***FISH*** values. Passed by reference.
- Matrix: 2D matrix of floating point values.
- Map: Associative array, keys can be integers, strings, or doubles.
- Structure: Experimental user defined aggregate type. For future expansion.

FISH Data Types – data.f3dat

Edit data.f3dat

```
1  model new
2  zone create brick
3  ; Data types - variable types do not need to be declared
4  fish define data
5      i = 0                          ; Integer
6      f = 3.4                        ; Real
7      s = 'Hello world!'             ; String
8      b = true                       ; Boolean
9      p = zone.head                  ; Pointer
10     v = vector(3,4,5)              ; Vector
11     t = zone.stress(p)             ; Tensor
12     a = array.create(100)          ; Array
13     m = matrix(4,4)                ; Matrix
14     n = map('fred',1,'george',2)   ; Map
15 end
16 @data
17
18 fish define fred
19     f = 'change type!'
20     i = float(0.0)             ; Force type
21 end
22 @fred
23
24 fish list contents @n ;; See contents of aggregate types
25 fish list contents @m
26
```

FISH Exercise – Parameterize a model

- Set up a FISH function that defined bulk and shear properties as parameters
- Create a 5x5x5 brick and load due to gravity

Edit parameter.f3dat

```
model new
; Model parameters
[bulk = 1e8]
[shear = 0.3e8]

zone create brick size 5 5 5
zone face skin
zone cmodel assign elastic
zone property density 2000 bulk @bulk shear @shear
model gravity 10
zone face apply velocity-normal 0 range group 'East' or 'West'
zone face apply velocity-normal 0 range group 'North' or 'South'
zone face apply velocity-normal 0 range group 'Bottom'
zone initialize-stresses ratio 0.5
model solve

```

FISH Vectors

Edit vectors.f3dat

```
model new
zone create brick
fish define dovectors
    local v2 = vector(1,2)    ;; 2D or
    local v3 = vector(3,4,5) ;; 3D vectors

    local v5 = v3 + v3        ;; Standard arithmetic operators apply, + - / * etc.
    local v6 = v3 / v5        ;; * / do component by component multiply
    local v7 = v5 * 4.0       ;; Can also mix / * operators with real

    local x = v3->x           ;; -> can be used to access components
    v5->z = 9                 ;; For writing as well

    local v8 = math.cross(v3,v5) ;; Lots of math support
    local f = math.dot(v5,v6)
    local f2 = math.mag(v5)

    local zone = zone.head
    local v = zone.pos(zone) ;; Lots of intrinsic functions return vectors
    x = zone.pos.x(zone)      ;; Almost all of them have component versions
    x = zone.pos(zone,1)
    ; x = zone.pos(zone)->x  ;; ERROR, -> can only be used on symbols
end
```

FISH Tensors

Edit tensors.f3dat

```
model new
zone create brick
fish define dotensors
    local t = tensor(1,2,3,4,5,6) ;; xx, yy, zz, xy, xz, yz
    local u = tensor(0,0,0)       ;; Three components required (if reals), all others optional

    local v = t * vector(1,2,3)   ;; tensor * vector support, as is tensor * real
    local w = t + t               ;; + - * / supported, all component by component
    local x = t * t

    x = t->xx          ;; -> can be used to access components
    t->zz = 9          ;; For writing as well

    dircos = array.create(3)  ; Arrays explained fully later
    v = tensor.prin(t,dircos) ; Get principal values and directions
    v->x = 3
    t = tensor.prin.from(v,dircos) ; Get full tensor from components and values

    local zone = zone.head
    t = zone.stress(zone)        ;; Lots of intrinsic functions return tensors
    x = zone.stress.xx(zone)     ;; Almost all of them have component versions
    x = zone.stress(zone,1,1)
    ; x = zone.stress(zone)->xx  ;; ERROR, -> can only be used on symbols
end
@dotensors
```

FISH Arrays

Edit arrays.f3dat

```
1  model new
2  fish define doarrays(n)
3      array fred(10)  ;; Array declaration, done when interpreted.
4                      ;; 10 must be a constant when interpreted.
5      ;; array fred(n)   ERROR
6
17/FLAC3D Training Course Work In Progress/Day4_FISH/intrinsics.f3dat -- Opened
8      global mary = array.create(3,3,3) ;; Can be arbitrarily dimensioned
9
10     george(4) = 'this is a test'
11     mary(1,2,3) = vector(4,5,6)   ;; Array entries can be any valid FISH value
12
13     local a = george    ;; george and a now REFER TO THE SAME ARRAY
14     a(4) = 'Different'
15
16     array.delete(mary)  ;; Generally want to clean up, but not absolutely necessary
17 end
18 @doarrays(100)
19
20 fish list @george(4) ;; Can list entries
21 fish list contents @george
22
```

FISH Matrices

Edit matrices.f3dat

```
model new
fish define domatrix
    local m = matrix(3,3)  ;; 1D or 2D matrices of real values
    local n = matrix(3,3)
    m(1,1) = 11
    m(2,2) = 22
    m(3,3) = 33
    m(1,2) = 12
    m(1,3) = 13
    m(2,1) = 12
    m(3,1) = 13
    m(2,3) = 23
    m(3,2) = 23
    local mi = matrix.inverse(m) ;; Matrix operations supported

    global i = m*mi              ;; Matrix multiplication supported

    local t = tensor(m)          ;; Can convert from tensor to matrix and vice versa
    local mt = matrix(t)

    local v = vector(1,2,3)
    m1 = matrix(v)           ;; Matrix from vector
    v = vector(m1)           ;; Vector from 3x1 matrix
    io.out(string(v))
end
@domatrix
fish list contents @i
list fish int 'matrix.'
```

FISH Operators

- Most operators work as expected: + - * /
  - a = b + c
  - e = f / g Warning: Integer division is not automatically promoted to float.
  - - can be used as a unary prefix: b = a * (-b)
- The = assignment operator is a bit special, in that only one can appear on a line
- ^ is used as an exponent/power operator. a^b means a^b.
- % is a modulus operator, 'a % b' is remainder of a/b. So 4%3 is 1. Also works with real values.
- == < > # >= <= are all testing operators returning a Boolean. Can use = too in if statement.
- & | ~ AND OR and NOT Boolean operators
- -> Member access operator, use with vectors, tensors, etc.
- Operator precedent order: ^ (unary)- / * % - + == > < # >= <= & |

FISH Intrinsic Functions

- At last count ***FLAC3D*** had 1,470 built in intrinsic functions.
  - 2,196 with ***PFC3D*** loaded.
- Help has master FISH intrinsic index
- CTRL-SPACE to look up
- F1 Context Help
- CAN execute alone on line, no = required

Edit intrinsics.f3dat

```
1  model new
2  fish define fred
3      local zone = zone.head                  ; Presence of . character identifies it as an intrinsic
4      local t = zone.stress.effective(zone)   ; Abbreviations are ok in all by last word as long as they are unambiguous
5            t = z.s.effective(zone)           ; These two refer to the same intrinsic
6                                              ; If you get it wrong it will be highlighted/.
7      ; CTRL-SPACE
8      ; F1
9      a = math.sinh
10 end
11 list fish int 'math.'
12
13
```

FISH Statements

• **local** – **global**	Create symbols
• **if** – **else if** – **else** – **endif**	Conditional branching
• **loop** – **endloop**	Looping
• **continue** – **exit loop**	Loop control
• **caseof** – **case** – **endcase**	Branching to multiple blocks
• **section** – **endsection** – **exit section**	Start and end a section of code
• **command** – **endcommand**	Execute ***FLAC3D*** commands
• **exit**	Exit function

- Alternate forms: **end_if**, **end_case**, **case_of**, **end_section**, **end_command**

FISH Conditionals

- **if** *expr* **then** ; ← **then** is optional
- **else if** expr **then**
- **else**
- **endif**

- *expr* is any expression that evaluates to a Boolean or an Integer (0 = false).

Edit conditional.f3dat

```
model new
fish define conditional
    local a = 0
    local b = 1
    local c = 3.4
    local t = true
    local f = false
    if a == 0 then      ; == comparison operator returning bool
        io.out('First')
    endif
    if c-3.4 then           ; Returns real value 0.0, evaluated as false
        io.out('Second')
    endif
    if a = 0 then           ; = can be used, special case
        io.out('Third')
    endif
    if t then               ; t is boolean
        io.out('Fourth')
    endif
    if t & (~f) then        ; Boolean expression!
        io.out('Fifth')
    endif
end
@conditional
```

FISH Looping

- Four forms of LOOP
  - **loop** *var* (*start*,*end*,*<inc>*)
    …
    endloop
  - **loop while** *expr*
    …
    endloop
  - **loop for** (*initialize*,*test*,*modify*)
    …
    endloop
  - **loop foreach** *var expr*
    …
    endloop

FISH While Loop, For Loop

Edit whileloop.f3dat

```
model new
fish define whileloop
    a = 1
    ; Expression evaluating to boolean,
    ;       integer, or float (0=false)
    loop while a < 5
        io.out(string(a))
        a = a + 1
    endloop
end
@whileloop
```

Edit forloop.f3dat

```
model new
fish define forloop
    loop for (local i=1,i<5,i=i+1)
        io.out(string(i))
    endloop
    io.out('Final '+string(i))

    ; Expressions can be anything
    loop for (local a='+',string.len(a)<5,a=a+'=')
        io.out(a)
    endloop
    io.out('Final '+a)
end
@forloop
```

FISH Foreach loop, loop control

Edit foreachloop.f3dat

```
model new
zone create brick size 2 2 2
fish define foreachloop
    ; Classing foreach loop
    loop foreach local zone zone.list ; Expression resulting in pointer to list
        io.out(string( zone.pos(zone)))
    endloop

    ; Loop control with CONTINUE, EXIT LOOP
    loop foreach zone zone.list
        if zone.id(zone)==2 then
            continue ; skips to next entry in list
        endif
        if zone.id(zone)==6 then
            exit loop    ; Exits loop completely
        endif
        io.out('ID = '+string(zone.id(zone)))
    endloop

    ; NOTE:  Deleting objects in the list is safe.
end
@foreachloop

```

FISH Section – Exit Section – EndSection

- Method of jumping forward in a function
- Skip lots of code, without creating deeply nested if statements

Edit section.f3dat

```
1  model new
2  fish define dosection(x)
3      section
4          io.out('One')
5          if x < 1 then
6              exit section
7          endif
8          io.out('Two')
9          if x < 3 then
10             exit section
11         endif
12         io.out('Three')
13     endsection
14     io.out('Final')
15 end
16 @dosection(4)
17 @dosection(2)
18 @dosection(0)
19
```

FISH Caseof – case - endcase

- Code branching faster and simpler than lots of else if statements

Edit caseof.f3dat

```
1  model new
2  fish define docaseof(n)
3      caseof  n
4      case 1
5          io.out('One')
6      case 2
7          io.out('Two')
8      case 3
9          io.out('Three')
10     case 4
11         io.out('Four')
12     case 'fred'
13         io.out('Fred')
14     case 'george'
15         io.out('George')
16     case 'mary'
17         io.out('Mary')
18     endcase
19 end
20 @docaseof(1)
21 @docaseof(2)
22 @docaseof(3)
23 @docaseof(4)
24 @docaseof('fred')   ; Comparison is case sensitive! (bug)
25 @docaseof('George')
26
```

FISH Command -- Endcommand

- ***FISH*** functions can actually execute commands
- Useful for things that don't have ***FISH*** function equivalents
- Can actually create new ***FISH*** functions……

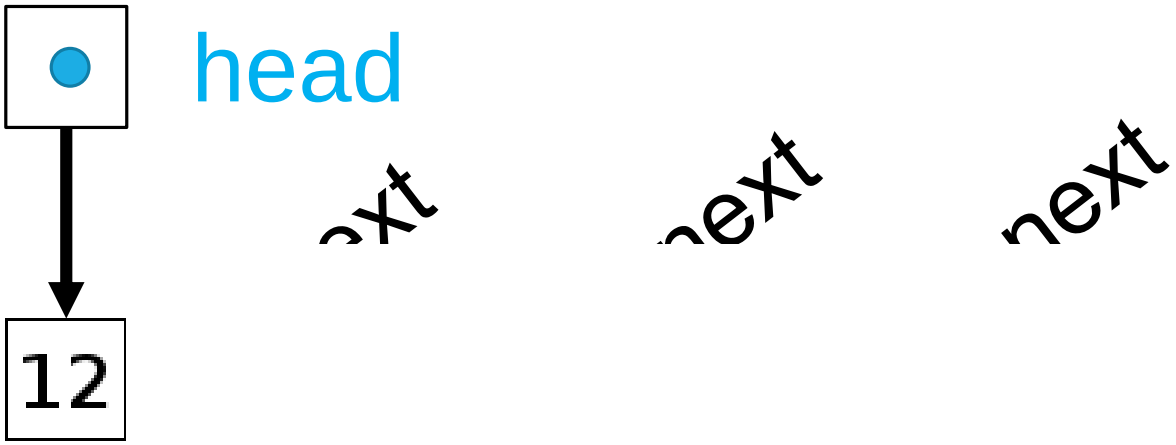
Edit commands.f3dat

```
model new
fish define docommand
    ; Create two bricks - note command/endcommand can see local values
    local v = vector(1,2,3)
    command
        zone create brick point 0 @v
        zone create brick point 0 [v+vector(10,0,0)]
    endcommand

    ; No zone.delete intrinsic, but can do with command/endcommand
    loop foreach local zone zone.list
        if zone.id(zone)%5==0 then
            command
                zone delete range id [zone.id(zone)]
            endcommand
        endif
    endloop

    ; Can execute a command string you build up yourself
    system.command('zone create brick point 0 '+string(v+vector(20,0,0)))
end
@docommand
```

Pointers and linked lists

Pointers store addresses to where in the memory other objects are

Linked lists are a clever and flexible way to use pointers to connect data



.L

FISH for *FLAC3D*: Zone and grid point functions

- All zones are contained in **`zone.list`**.
- All grid points contained in **`gp.list`**.
- *All* zones point to 8 grid points
  - Some some types will point to the same grid point multiple times.
- *All* zones have six face join values
  - Point to zone across that face
  - Always null if face is degenerate
- Use function on left to assign values: **`zone.stress`**`(zone) =` **`tensor`**`(1,2,3)`

Edit zone1.f3dat

```
1  model new
2  zone create brick size 2,2,2
3  fish define zone1  ;; Loop through all zones demonstration
4      loop foreach local zone zone.list
5          io.out(string(zone.id(zone)))
6      endloop
7      loop foreach local gp gp.list
8          io.out(string(gp.pos(gp)))
9      endloop
10 end
11 @zone1
12
13
```

- See Help Index, F1, Ctrl-Space for list of all zone and gp intrinsics!

FISH for ***FLAC3D***: Exercize – list grid point positions

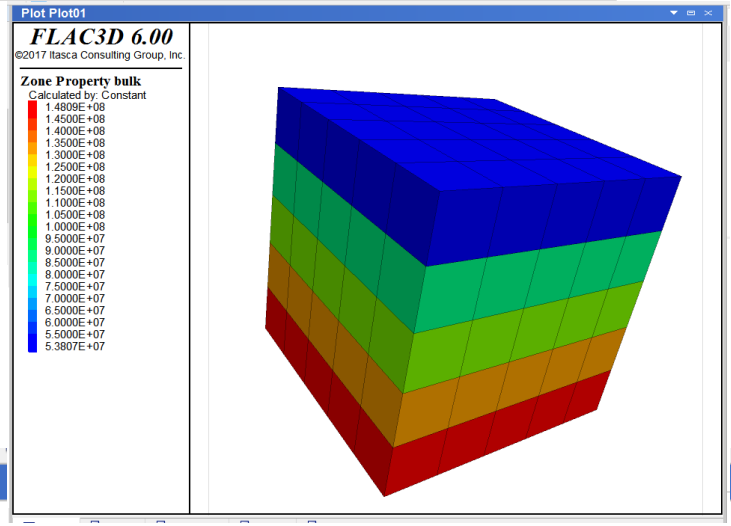
- Make a 5x5x5 mesh, and list the positions of all the grid points

Edit listpos.f3dat

```
1  model new
2  zone create brick size 5,5,5
3  fish define listpos
4      loop foreach local gp gp.list
5          io.out('id '+string(gp.id(gp))+' is at '+string(gp.pos(gp)))
6      endloop
7  end
8  @listpos
9
10
```

FISH for ***FLAC3D***: Exercize – Vary properties

- Make a 5x5x5 mesh
- Assign elastic material model
- Vary bulk and shear modulus with depth:
  - $K = \frac{2}{3}(E_0 + c\sqrt{depth})$
  - $G = \frac{2}{5}(E_0 + c\sqrt{depth})$
  - $E_0 = 1e7$
  - $c = 1e8$



Edit varyprop.f3dat

```
1  model new
2  zone create brick size 5,5,5
3  zone cmodel assign elastic
4  fish define varyprop
5      loop foreach local zone zone.list
6          local depth = 5.0 - zone.pos.z(zone)
7          local value = 2.0 * (1e7 + 1e8*math.sqrt(depth))
8          zone.prop(zone,'bulk')  = value / 3.0
9          zone.prop(zone,'shear') = value / 5.0
10     endloop
11 end
12 @varyprop
13
```

FISH for *FLAC3D*: Extra Variables

- Each zone, grid point, and face can store ***FISH*** values.
  - Any valid FISH value can be stored in each entry.
  - Up to 128 in each object – be aware this uses memory!
- `zone.extra(zone,index)`
- `gp.extra(zone,index)`
- `zone.face.extra(zone,face,index)`

- For example:

```
zone.extra(zone,3) = vector(3,4,5)
local v = zone.extra(zone,3)
```

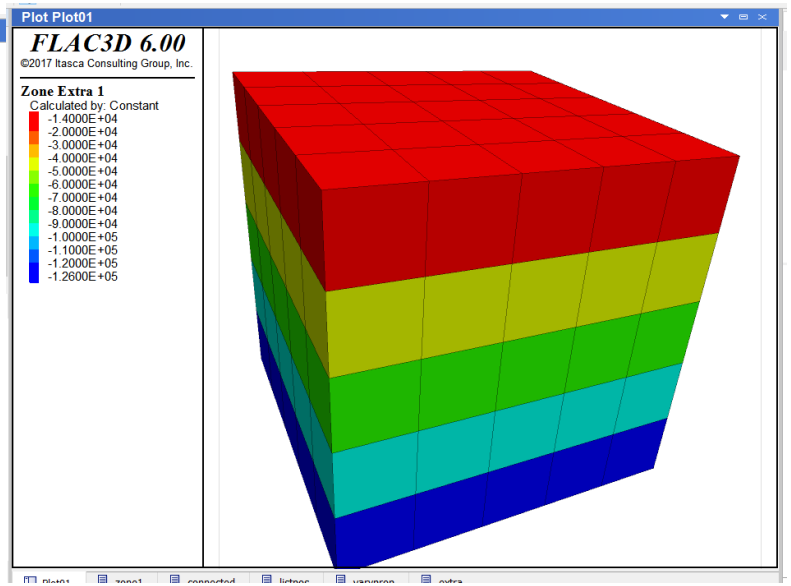
- Extra values can be contoured: as scalars, vectors, or tensors

FISH for *FLAC3D*: Exercise – Plot Custom Value

- Start with ‘varyprop’ model
- Add boundary conditions and reach initial equilibrium
- Calculate (szz + 2.0 / 5.0 * sxx) and store in extra 1.

Edit extra.f3dat

```
model restore 'varyprop'
; Boundary conditions
zone face skin
zone face apply velocity-normal 0 range group 'East' or 'West'
zone face apply velocity-normal 0 range group 'North' or 'South'
zone face apply velocity-normal 0 range group 'Bottom'
; Initial conditions
zone property density 2000
model gravity 10
zone initialize-stresses
model solve
;
fish define extra
    loop foreach local zone zone.list
        local str = zone.stress(zone)
        zone.extra(zone,1) = str->zz + (2.0/5.0)*str->xx
    endloop
end
@extra
mode save 'extra'
```



FISH for ***FLAC3D***: Structural Element Data

- Every structural element points to 2 or 3 nodes: **`struct.node`**`(elem,ind)`
- Every node has 1 or 2 links: **`struct.node.link`**`(node,<ind>)`
- All structural elements (of all six types: beams, cables, piles, shells, liners, geogrids) are stored in one global list: **`struct.list`**
- Structural nodes and links are stored in their own global list:
  - **`struct.node.list`**
  - **`struct.link.list`**
- Some element types share functions:
  - Pointer to all six types of elements can be used with **`struct.`** functions.
  - Pointers to Piles can be used with **`struct.beam.`** functions.
  - Pointers Geogrid and Liners can be used with **`struct.shell.`** functions.
- Note that stress resultants and stresses must be calculated on the command line!

FISH for *FLAC3D*: Histories

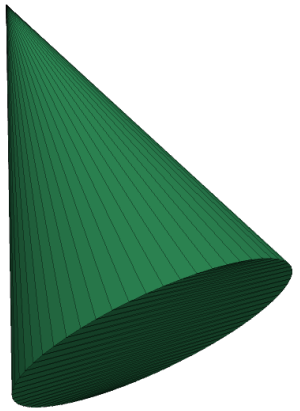
- ***FISH*** functions can be executed and their (real) return values stored as histories.
- The `fish history` command:
  - `fish history name='fred' @fred`
- The symbol may not be a function, if so the current value is retrieved.
- History values are retrieved in the order they are declared.
  - So a function taken first can set global values that are taken later.

FISH for ***FLAC3D***: Geometry, Extruder, B-Blocks, etc

- The Geometry, Extruder, and Building-Blocks logic stores data in named “sets”.
- Each set has it’s own list of polygons, edges, nodes, etc.
- So functions need to pass set pointers in:
- For example, the following function lists the centroid of every polygon in a geometric set:

Edit geometry.f3dat

```
model new
geometry select 'Fred'
geometry generate cone base (0,0,0) axis (0,0,1) ...
& height 10 radius 3 resolution 0.1
;
fish define listpoly(name)
    local set = geom.set.find(name)
    loop foreach local poly geom.poly.list(set)
        io.out(string(geom.poly.pos(poly)))
    endloop
end
@listpoly('Fred')
```



FISH Further Topics: File I/O

- FISH can read and write to files.
  - **file.open**
  - **file.close**
  - **file.read**
  - **file.write**
  - **file.read.pointer**
  - **file.pos**
- There are some string functions that are typically handy:
  - **string.token**
  - **string.token.type**
- **file.open**(name,opentype,file type)
  - Returns integer, 0 = no error
  - opentype = 0: read
    = 1: write and overwrite
    = 2: write and append
  - filetype = 0: FISH mode
    = 1: String mode
    = 2: Binary mode
  - This refers to a single global file.

FISH Further Topics: File I/O - reading

- **`file.read`** behavior depends on argument types and number.
- Simplest: **`file.read`**`(arr,10)`
  - Reads 10 units from the global file and places them into array `arr`, assumed to be a one dimensional array.
  - Reads ***FISH*** values in ***FISH*** mode, lines as strings in String mode, and bytes as integers in Binary mode.
- Pointer: **`file.read`**`(arr,10,filepnt)`
  - Same as above, but reads from then file indicated by `filepnt` returned from **`file.open.pointer`**.
- ReadAll: **`file.read`**`(name,openmode,arr,num,<pos>)`
  - This version opens the file `name` and reads `num` units (based on `openmode`, as in **`file.open`**) into array `arr`, then closes the file.
  - `pos`, if given, indicates the file position (in bytes) to start at.
  - Handy to read an entire file contents at once into a large array.

FISH Further Topics: File I/O - writing

- **`file.write`** behaves the same as **`file.read`**.
- Simplest: **`file.write(`**`arr,10)`
  - Writes 10 units from the global file and places them into array `arr`, assumed to be a one dimensional array.
  - Writes ***FISH*** values in ***FISH*** mode and , lines as strings in String mode, and bytes as integers in Binary mode.
- Pointer: **`file.write`**`(arr,10,filepnt)`
  - Same as above, but writes from then file indicated by `filepnt` returned from **`file.open.pointer`**.
- Read All: **`file.write`**`(name,openmode,arr,num,<pos>)`
  - This version opens the file `name` and writes `num` units (based on `openmode`, as in **`file.open`**) into array `arr`, then closes the file.
  - `pos`, if given, indicates the file position (in bytes) to start at.
  - Handy to write an entire array into a file at once.

FISH Further Topics: File I/O - example

Edit file1.f3dat

```
model new
fish define setup
    global arr = array.create(6)
    arr(1) = 10
    arr(2) = 3.4
    arr(3) = vector(5,6)
    arr(4) = vector(7,8,9)
    arr(5) = 'this is a test'
    arr(6) = 'of the system'
    global two = array.create(6)
end
@setup
```

```
fish define fred
    file.open('test.fis',1,0)  ; 1 = write and overwrite, 0 = FISH mode
    file.write(arr,6)
    file.close

    file.open('test.fis',0,0)  ; 0 = read, 0 = FISH mode
    file.read(two,6)
    file.close

    ;; Is equivalent to
    local fpnt = file.open.pointer('test.fis',1,0)
    file.write(arr,6,fpnt)
    file.close(fpnt)

    fpnt = file.open.pointer('test.fis',0,0)
    file.read(two,6,fpnt)
    file.close(fpnt)

    ;; Is equivalent to
    file.write('test.fis',0,arr,6)

    file.read('test.fis',0,two,6)
end
@fred
```

FISH Further Topics: File I/O - tokens

- **`string.token`**`(str,i)`
- Converts string `str` into ‘tokens’, using same rules as command processor.
- Spaces, tabs, parentheses (), commas, equals are all token delimiters.
- Returns contents at i^{th} token, interpreted as parameter (integer, real, or string).
- So the string `'Fred has 6 silly monkeys with 5.5 hairs'` will have:
  - 8 tokens
  - Tokens 1,2,4,5,6,8 will return strings `'Fred'`, `'has'`, `'silly'`, `'monkeys'`, `'with'`, and `'hairs'`
  - Token 3 will return the integer 6
  - Token 7 will return the real 5.5
- **`string.token.type`**`(str,i)` uses the same rules but returns type
  - So Tokens 1,2,4,5,6,8 will return 3 = string
  - Token 3 will return 1 = integer
  - Token 7 will return 2 = real

FISH Further Topics: File I/O - example

```
14 fish define fred
15     file.open('test.fis',1,1)  ; 1 = write and overwrite, 0 = String mode
16     file.write(arr,6)
17     file.close
18
19     file.open('test.fis',0,1)  ; 0 = read, 0 = String mode
20     file.read(two,6)
21     file.close
22
23     ;; Is equivalent to
24     local fpnt = file.open.pointer('test.fis',1,1)
25     file.write(arr,6,fpnt)
26     file.close(fpnt)
27
28     fpnt = file.open.pointer('test.fis',0,1)
29     file.read(two,6,fpnt)
30     file.close(fpnt)
31
32     ;; Is equivalent to
33     file.write('test.fis',1,arr,6)
34
35     file.read('test.fis',1,two,6)
36
37     io.out(string.token(two(6),3))
38 end
39 @fred
40 list fish contents @two
```

```
flac3d>@fred
system
flac3d>list fish contents @two
[ '10' '3.4' '(5,6)' '(7,8,9)' 'this is a test' 'of the system' ]
```

FISH Further Topics: Callbacks while cycling

- ***FISH*** histories are an example of ***FISH*** calls during cycling.
- `fish callback`
- The `fish` and `fish-local` modifiers to `zone face apply` and `zone apply`.
- `model solve fish-call` and `model solve fish-halt`.

FISH Further Topics: Callbacks while cycling

- ***FISH*** histories are an example of ***FISH*** calls during cycling.
- `fish callback`
- The `fish` and `fish-local` modifiers to `zone face apply` and `zone apply`.
- `model solve fish-call` and `model solve fish-halt`.

FISH Further Topics: Callbacks while cycling

- `fish callback add @name t <keyword>`
- The fish function `name` is called at time index `t` in the cycle sequence (listed at right).
- Can specify `interval n` so it is only called every `n` steps.
- Can specify `process fluid` (for example) so it is only called when fluid is active.
- Can use a floating point time index to insert yourself at any point in the cycle index.

-10	Validate zones
-9.5	Validate structural element dynamic
-2	Validate things that couple to zones
-1	Validate zone dynamic
0	Calculate timestep
20	Increment time
39	Zero zone grid point values
41	Calculate zone stress increments from strain increments
41.15	Structural element initialize node values
41.16	Structural element constitutive first
41.17	Structural element constitutive second
60.5	Zone attach send forces
61	Zone grid point equations of motion
61.1	Read velocities from attach conditions
61.15	Structural element equation of motion
61.2	Zone thermal coupling
62	Zone thermal calculations
81	Zone fluid flow calculations
81.1	Zone stress coupling to fluid flow
81.2	Zone fluid particle tracking
100	Calculate zone convergence criteria
101	Calculate structural element convergence criteria

FISH Further Topics: zone face apply callbacks

- A `fish @name` multiplier can be specified.
- The function is called every cycle, and the value returned is multiplied by the base value given to the apply command.

```
22 fish define wave
23     wave = 0.5 * (1.0 - math.cos(8.0*math.pi*math.min(zone.dynamic.time.total,0.25)))
24 end
25 zone face apply stress-xz -2e5 fish @wave range group 'Bottom'
```

FISH Further Topics: zone field data

- Zone data can be calculated ***at any location*** inside a zone.

Edit field.f3dat

```
model restore 'base'
fish define getszz
    global data = array.create(100)
    zone.field.name = 'stress'
    zone.field.quantity = 'xx'
    zone.field.method.name = 'polynomial'
    zone.field.init   ; Optimizes for multiple calls with same value and method
    loop local i (1,100)
        local z = 5.0 + float(ii)/10.0
        data(i) = zone.field.get(3.5,4.3,z)
    endloop
end
@getszz
```