Engineering Sketch Pad (ESP)

User Training Session 3
CSM Language

John F. Dannenhoffer, III
john@geocentrictech.com
Geocentric Technologies LLC

updated for v1.28

Overview (1)

- Format of `.csm` file
- Special characters
- Numbers
- Parameters
  - Types
  - Names
  - Dimensions
  - Lower and Upper Bounds
- Expressions
  - Numeric
  - String

Overview (2)

- Looping
  - PATBEG, PATBREAK, PATEND
- Logic
  - IFTHEN, ELSEIF, ELSE, ENDIF
- Signal Handling
  - THROW, CATBEG, CATEND
- Reading Help File
- Homework Exercises

Opening Thoughts

- All configuration information is contained in `.csm` (or possibly `.udc`) files
  - `.csm` files are plain ASCII text that are readable by humans
  - because they are ASCII files, they can either be written directly by humans (using any text editor) or by other programs
- When you build a configuration using the `ESP` user interface, you are actually building a `.csm` file
- Using the interface can be effective for beginning users who are building small models
- Once a user gets experience with `ESP`, most of the models are created by "typing" a `.csm` directly

Format of the `.csm` file (1)

- The `.csm` file contains a series of statements.
- If a line contains a hash (#), all characters starting at the hash are ignored.
- If a line contains a backslash (\), all characters starting at the backslash are ignored and the next line is appended; spaces at the beginning of the next line are treated normally.
- All statements begin with a keyword (described below) and must contain at least the indicated number of arguments.
- The keywords may either be all lowercase or all UPPERCASE (but not MixedCase).
- Any CSM statement can be used in a `.csm` file except the `INTERFACE` statement.

Format of the `.csm` file (2)

- Blocks of statements must be properly nested. The Blocks are bounded by
  - `PATBEG/PATEND`
  - `IFTHEN/ELSEIF/ELSE/ENDIF`
  - `SKBEG/SKEND`
  - `SOLBEG/SOLEND`
  - `CATBEG/CATEND`
- Extra arguments in a statement are discarded. If one wants to add a comment, it is recommended to begin it with a hash (#) in case optional arguments are added in future releases.
- Any statements after an `END` statement are ignored.
  - hint: if debugging, consider THROWing an error instead to avoid unclosed Blocks
- All arguments must not contain any spaces or must be enclosed in a pair of double quotes (for example, `"a + b"`).

Format of the `.csm` file (3)

- Parameters are evaluated in the order that they appear in the file, using MATLAB-like syntax (see 'Expression rules' below).
- During the build process, `OpenCSM` maintains a last-in-first-out (LIFO) "Stack" that can contain Bodys, Marks, and Sketches.
- The `.csm` statements are executed in a stack-like way, taking their inputs from the Stack and depositing their results onto the Stack.
- The default name for each Branch is `Brch_xxxxxx`, where `xxxxxx` is a unique sequence number.

Special characters (1)

```
#               introduces comment
"               ignore spaces until following "
\               ignore this and following characters and
                   concatenate next line
<space>         separates arguments in .csm file (except
                   between " and ")
0-9             digits used in numbers and in names
A-Z a-z         letters used in names
_ : @           characters used in names (see rule for names)
.               decimal separator (used in numbers),
                   introduces dot-suffixes (in names)
,               separates function arguments and row/column
                   in subscripts
;               multi-value item separator
```

Special characters (2)

```
( )          groups expressions and function arguments
[ ]          specifies subscripts in form [row,column] or [index]
{ } < > ~    characters used in strings
+ - * / ^    arithmetic operators
$            as first character, introduces a string that is
                terminated by end-of-line or un-escaped plus,
                comma, or close-parenthesis
@            as first character, introduces @-parameters
'            used to escape comma, plus, or close-parenthesis
                within strings
!            if first character of implicit string, ignore
                $! and treat as an expression

|            cannot be used (reserved for OpenCSM internals)
&            cannot be used (reserved for OpenCSM internals)
```

Numbers

- Start with a digit or decimal (.)
- Followed by zero or more digits and/or decimals (.)
- There can be at most one decimal in a number
- Optionally followed by an e, e+, e-, E, E+, or E-
- If there is an e or E, it must be followed by one or more digits
- If numbers are in a list, the elements are separated by a semicolon (;)

Types of Parameters (1)

- Design Parameter
  - values are declared in a `DESPMTR` statement
    - in `.csm` file or
    - in top-level include-type `.udc` file
  - must contain one or more numbers (no strings)
  - if multi-valued, must be first `DIMENSION`ed
  - can contain lower- and upper-bounds, specified in `LBOUND` and `UBOUND` statements
  - values are only visible at the top-level
  - values can be changed by a call to `ocsmSetValu` or `ocsmSetValuD` (after `ocsmLoad` and before `ocsmBuild`)
  - values can be read by call to `ocsmGetValu`
  - sensitivities can be computed by a call to `ocsmSetVel` or `ocsmSetVelD`

Types of Parameters (2)

- Configuration Parameter
  - values are declared in a `CFGPMTR` statement
    - in `.csm` file or
    - in top-level include-type `.udc` file
  - must contain one or more numbers (no strings)
  - if multi-valued, must be first `DIMENSION`ed
  - can contain lower- and upper-bounds, specified in `LBOUND` and `UBOUND` statements
  - values are only visible at the top-level
  - values can be changed by a call to `ocsmSetValu` or `ocsmSetValuD` (after `ocsmLoad` and before `ocsmBuild`)
  - values can be read by call to `ocsmGetValu`
  - sensitivities CANNOT be computed for Configuration Parameters

Types of Parameters (3)

- Constant Parameter
  - values are declared in a `CONPMTR` statement
    - in `.csm` file
    - in top-level include-type `.udc` file
  - must contain only one number (no strings)
  - values are visible from any `.csm` or `.udc` file
  - values CANNOT be changed by a call to `ocsmSetValu` or `ocsmSetValuD`
  - sensitivities CANNOT be computed for Constant Parameters

Types of Parameters (4)

- Local Variables
  - is created by a `SET`, `PATBEG` or `GETATTR` statement
  - can contain one or more numbers or a character string
  - if multi-valued, must first be `DIMENSION`ed
  - can be an @-parameter (described below)
  - are only usable in `.csm` or `.udc` file in which it was defined (unless the `.udc` file has `INTERFACE . ALL` in its preamble)
- Output Parameters
  - declared in a `OUTPMTR` statement
  - refers to any local variable whose value is available outside `ESP` (such as to `CAPS`)

Parameter Type Summary

	DESPMTR	CFGPMTR	CONPMTR	OUTPMTR	LOCALVAR
Can be vector or array of numbers	Y	Y	N	Y	Y
Can have a string value	N	N	N	Y	Y
Can be restricted by `LBOUND` or `UBOUND`	Y	Y	N	N	N
Scope	T	T	G	L	L
Defined during `ocsmLoad` or `ocsmLoadDict`	Y	Y	Y	N	N
Can be set via `ocsmSetValu(D)`	Y	Y	N	N	N
Defined and set during `ocsmBuild`	N	N	N	Y	Y
Can be read via `ocsmGetValu(S)`	Y	Y	Y	Y	Y*
Can find associated sensitivity	Y	N	N	N	N

Y*=Parameter index may be different for different builds

scopes: T=top-level, G=global, L=local

Notes on the `DIMENSION` Statement

- General form is: `DIMENSION $pmtrName nrow ncol`
- Can only be applied once to a `DESPMTR` or `CFGPMTR`
- Cannot be applied to a `CONPMTR`
- When applied to an `OUTPMTR` or `LOCALVAR`
  - if the new size has fewer elements than the old size
    - the old values are copied to fill the new size
    - extra old elements are lost
  - if the new size has more elements than the old size
    - the old values are all copied
    - the last old value is copied into all the remaining new locations

Valid Parameter Names

- Start with a letter, colon (:), or at-sign (@)
- Contains letters, digits, at-signs (@), underscores (_), and colons (:)
- Contains fewer than 64 characters
- Names that start with an at-sign cannot be set by a `CONPMTR`, `DESPMTR`, `CFGPMTR`, `SET`, `PATBEG`, or `GETATTR` statement
- When listed in the `ESP` user interface, parameters are sub-grouped based upon the colons (:)

Dot-suffixes (1)

- If a name has a dot-suffix, a property of the parameter (and not its value) is returned

`x.nrow`	number of rows in `x` (0 for string)
`x.ncol`	number of columns in `x` (0 for string)
`x.size`	number of elements or characters in `x`
`x.sum`	sum of elements in `x`
`x.norm`	RMS norm of elements in `x`
`x.min`	minimum value in `x`
`x.max`	maximum value in `x`

Dot-suffixes (2)

- Example:

```
DIMENSION  myvar  2 3
DESPMTR    myvar  "1; 2; 3;\
                   4; 5; 6"
```

  - `myvar.nrow` returns 2
  - `myvar.sum` returns 21

- To make a copy of array A, use:

```
DIMENSION copyOfA A.nrow A.ncol
SET       copyOfA A
```

Accessing Element of an Array

- Basic format is: `name[irow,icol]` or `name[ielem]`
- Name must follow rules above
- `irow`, `icol`, and `ielem` must be valid (integer) expressions
- `irow`, `icol`, and `ielem` start counting at 1
- For 2D arrays, either `name[irow,icol]` or `name[ielem]` be used
- Values are stored across rows ([1,1], [1,2], ..., [2,1], ...)

@-parameters (1)

- Every time a Body gets created, or after a `SELECT` statement, readable local variables are set

@-parameters (2)

```
        body face edge node  <- last SELECT
@seltype -1    2    1    0   selection type
@selbody  x    -    -    -   current Body
@sellist -1    x    x    x   list of Nodes/Edges/Faces

@nbody    x    x    x    x   number of Bodys
@ibody    x    x    x    x   current   Body
@nface    x    x    x    x   number of Faces in @ibody
@iface   -1    x   -1   -1   current   Face  in @ibody
@nedge    x    x    x    x   number of Edges in @ibody
@iedge   -1   -1    x   -1   current   Edge  in @ibody
@nnode    x    x    x    x   number of Nodes in @ibody
@inode   -1   -1   -1    x   current   Node  in @ibody
@igroup   x    x    x    x   group of current Body
@itype    x    x    x    x   0=NodeBody, 1=WireBody,
                                        2=SheetBody, 3=SolidBody
@nbors   -1    x    -    x   number of incident Edges
@nbors   -1    -    x    -   number of incident Faces
```

@-parameters (3)

```
@ibody1 -1    x    x   -1   first  element of 'Body' Attribute in @ibody
@ibody2 -1    x    x   -1   second element of 'Body' Attribute in @ibody

@xmin    x    x    *    x   x-min of bounding box or x at beg of edge
@ymin    x    x    *    x   y-min of bounding box or y at beg of edge
@zmin    x    x    *    x   z-min of bounding box or z at beg of edge
@xmax    x    x    *    x   x-max of bounding box or x at end of edge
@ymax    x    x    *    x   y-max of bounding box or y at end of edge
@zmax    x    x    *    x   z-max of bounding box or z at end of edge

@length  0    0    x    0   length of edge
@area    x    x    0    0   area of face or surface area of body
@volume  x    0    0    0   volume of body (if a solid)

@xcg     x    x    x    x   location of center of gravity
@ycg     x    x    x    x
@zcg     x    x    x    x
```

@-parameters (4)

```
@Ixx      x     x     x     0   centroidal moment of inertia
@Ixy      x     x     x     0
@Ixz      x     x     x     0
@Iyx      x     x     x     0
@Iyy      x     x     x     0
@Iyz      x     x     x     0
@Izx      x     x     x     0
@Izy      x     x     x     0
@Izz      x     x     x     0

@signal    x     x     x     x   current signal code
@nwarn     x     x     x     x   number of warnings

@edata                           only set up by EVALUATE statement
@stack                           Bodys on stack: 0=mark, -1=none
@scope                           scoping level (at last SELECT)
@version                         version number

in above table:
   x -> value is set
   - -> value is unchanged
   * -> special value is set (if edge)
   0 -> value is set to  0
  -1 -> value is set to -1
```

Expression Rules (Valid operators)

- Valid operators (in order of precedence):

( )	parentheses, inner-most evaluated first
func(a,b)	function arguments, then function itself
$\wedge$	exponentiation (evaluated left to right)
* /	multiply and divide (evaluated left to right)
+ -	add and subtract (evaluated left to right)

String Variables

- Contains the sequence of characters starting after a dollar-sign($) and ending with a space, plus-sign (+), comma (,), or closed-parenthesis ())
- If escaped with an apostrophe (’), can contain a plus-sign (’+), comma (’,) or closed-parenthesis (’))
  - for example:

    `$thisStringContainsAComma(’,’)`

    returns `thisStringContainsAComma(,)`
- Can never contain a space
- Are parsed left-to-right, as is any expression
  - for example:

```
SET    one 1
SET    mystr $thereIsA+one+$inThisString
```

    returns (in `mystr`) `thereIsA1inThisString`

Functions (1)

`pi(x)`	3.14159...*x
`min(x,y)`	minimum of x and y
`max(x,y)`	maximum of x and y
`sqrt(x)`	square root of x
`abs(x)`	absolute value of x
`int(x)`	integer part of x ($3.5 \rightarrow 3$, $-3.5 \rightarrow -3$)
	produces derivative=0
`nint(x)`	nearest integer to x
	produces derivative=0
`ceil(x)`	smallest integer not less than x
	produces derivative=0
`floor(x)`	largest integer not greater than x
	produces derivative=0

Functions (2)

`mod(a,b)`	modulus(a/b), with same sign as a and b$\geq$0
`sign(test)`	returns -1, 0, or +1
	produces derivative=0
`exp(x)`	exponential of x
`log(x)`	natural logarithm of x
`log10(x)`	common logarithm of x

Functions (3)

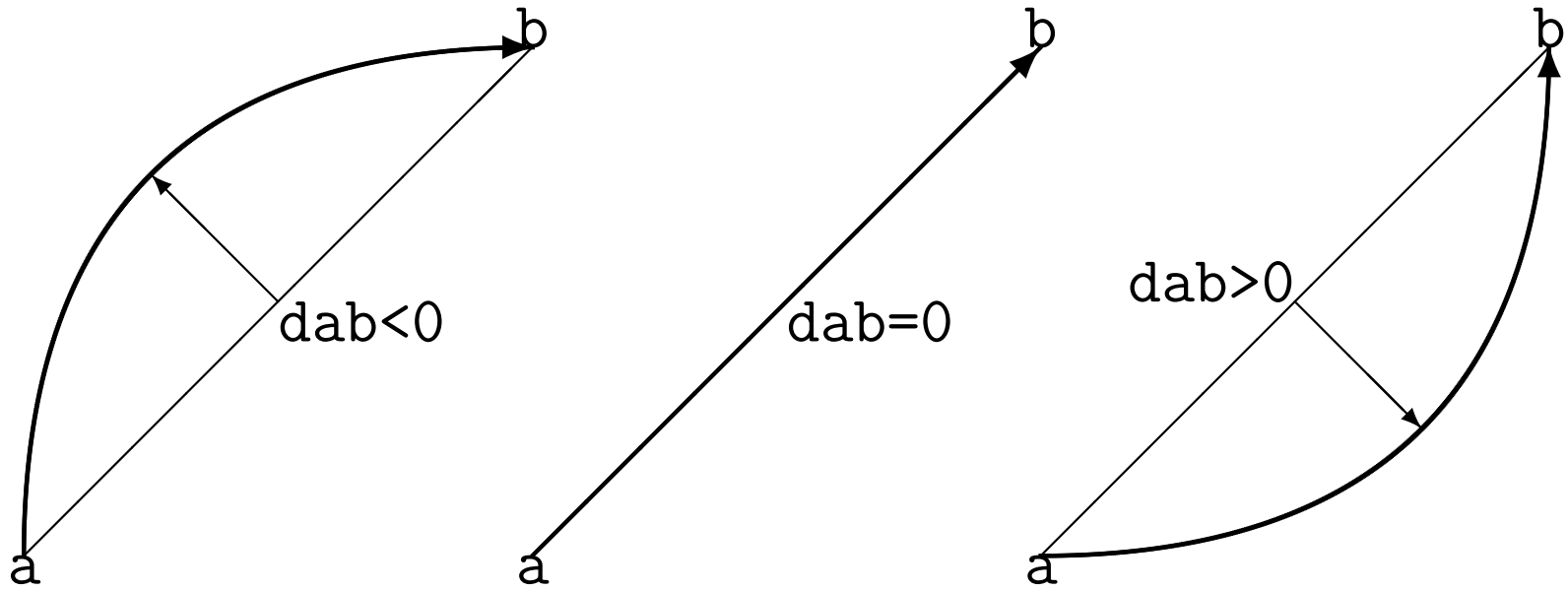
`sin(x)`	sine of x	(in radians)
`sind(x)`	sine of x	(in degrees)
`asin(x)`	arc-sine of x	(in radians)
`asind(x)`	arc-sine of x	(in degrees)
`cos(x)`	cosine of x	(in radians)
`cosd(x)`	cosine of x	(in degrees)
`acos(x)`	arc-cosine of x	(in radians)
`acosd(x)`	arc-cosine of x	(in degrees)

Functions (4)

`tan(x)`	tangent of x	(in radians)
`tand(x)`	tangent of x	(in degrees)
`atan(x)`	arc-tangent of x	(in radians)
`atand(x)`	arc-tangent of x	(in degrees)
`atan2(y,x)`	arc-tangent of y/x	(in radians)
`atan2d(y,x)`	arc-tangent of y/x	(in degrees)
`hypot(x,y)`	hypotenuse: $\sqrt{x^2+y^2}$	
`hypot3(x,y,z)`	hypotenuse: $\sqrt{x^2+y^2+z^2}$	

Circular Arc (Dip) Nomenclature

Arc going from point `a` to point `b`



Functions (5)

`Xcent(xa,ya,dab,xb,yb)`	X-center of circular arc produces derivative=0
`Ycent(xa,ya,dab,xb,yb)`	Y-center of circular arc produces derivative=0
`Xmidl(xa,ya,dab,xb,yb)`	X-point at midpoint of circular arc produces derivative=0
`Ymidl(xa,ya,dab,xb,yb)`	Y-point at midpoint of circular arc produces derivative=0
`seglen(xa,ya,dab,xb,yb)`	length of segment produces derivative=0

Functions (6)

`incline(xa,ya,dab,xb,yb)`	inclination of chord (in degrees) produces derivative=0
`radius(xa,ya,dab,xb,yb)`	radius of curvature (or 0 for linseg) produces derivative=0
`sweep(xa,ya,dab,xb,yb)`	sweep angle of circular arc (in degs) produces derivative=0
`turnang(xa,ya,dab,...` `  xb,yb,dbc,xc,yc)`	turning angle at b (in degrees) produces derivative=0
`dip(xa,ya,xb,yb,rad)`	acute dip between arc and chord produces derivative=0
`smallang(x)`	ensures $-180 \leq x \leq 180$

Functions (7)

`val2str(num,digits)`	convert `num` to a string
`str2val(string)`	convert `string` to a number
`findstr(str1,str2)`	finds location of `str2` in `str1` (bias-1) or 0 if not found
`slice(str,ibeg,iend)`	substring of `str` from `ibeg` to `iend` (bias-1)
`path($pwd)`	returns present working directory
`path($csm)`	returns directory of current `.csm` file
`path($root)`	returns $ESP_ROOT
`path($file)`	returns name of `.csm` file

Functions (8)

`ifzero(test,ifTrue,ifFalse)`	if `test` $= 0$, return `ifTrue`, else return `ifFalse`
`ifpos(test,ifTrue,ifFalse)`	if `test` > 0, return `ifTrue`, else return `ifFalse`
`ifneg(test,ifTrue,ifFalse)`	if `test` < 0, return `ifTrue`, else return `ifFalse`
`ifnan(test,ifTrue,ifFalse)`	if `test` is NaN, return `ifTrue`, else return `ifFalse`

Patterns (1)

- Patterns are like “for” or “do” loops
  - the Branches between the `PATBEG` and `PATEND` are executed a known number of times
  - at the beginning of each “instance”, the pattern number is incremented (from 1 to the number of copies)
  - one can break out of the pattern early with a `PATBREAK` statement
    - breaks out if expression evaluates to a non-zero value
  - patterns can be nested within other patterns

Patterns (2)

- Example pattern (indentation optional):

```
PATBEG      i     3
   SET      j     i-1
   BOX      j     0   0   1   1   1
   ROTATEX  j*10  0   0
PATEND
```

- is the same as:

```
BOX      0   0   0   1   1   1
ROTATEX  0   0   0

BOX      1   0   0   1   1   1
ROTATEX  10  0   0

BOX      2   0   0   1   1   1
ROTATEX  20  0   0
```

If/then (1)

- If/then constructs are used to make a choice within a `.csm` script
  - start with `IFTHEN` statement
  - has zero or more `ELSEIF` statements
  - has zero or one `ELSE` statement
  - has exactly one `ENDIF` statement
- The `IFTHEN` and `ELSEIF` statements have arguments, which can be specified in lowercase or UPPERCASE
  - `val1` — an expression
  - `op1` — can be `lt`, `le`, `eq`, `ge`, `gt`, `ne`, `LT`, ...
  - `val2` — an expression
  - `op2` — can be `or`, `xor`, `and`, `OR`, ... (defaults to `and`)
  - `val3` — an expression (defaults to `0`)
  - `op3` — can be `lt`, `le`, `eq`, `ge`, `gt`, `ne`, `LT`, or ... (defaults to `eq`)
  - `val4` — an expression (defaults to `0`)

If/then (2)

- Example (indentation optional):

```
IFTHEN    a   eq  4   or  b   ne  2
   BOX    0   0   0   1   1   1
ELSEIF    c   eq  sqrt(9)
   BOX    2   2   2   2   2   2
ELSE
   BOX    3   3   3   3   3   3
ENDIF
```

- Note that only one of the `BOX` commands will be executed

Throw/catch (1)

- Throw/catch constructs are used to generate and react to signals (errors)
- Signals can be generated by
  - executing a `THROW` command
    - `ESP` uses negative signal numbers, so users should generally use positive signal numbers to avoid collisions
  - a run-time error encountered elsewhere (see "help" for more info)
- When a signal is generated, all Branches are skipped until a matching `CATBEG` statement is encountered
  - the signal is cancelled
  - processing continues at the statement following the `CATBEG`
- If a `CATBEG` statement is encountered when there is no pending signal (or the pending signal does not match the `CATBEG`)
  - all Branches up to, and including the matching `CATEND` statement, are skipped

Throw/catch (2)

```
 1: BOX      0 0 0 1 1 1

 2: THROW    99

 3: SPHERE   0 0 0 1

 4: CATBEG   98
 5:    SPHERE    0 0 0 2
 6: CATEND

 7: SPHERE   0 0 0 3

 8: CATBEG   99
 9:    BOX         1 0 0 1 1 1
10: CATEND

11: CATBEG   99
12:    SPHERE   0 0 0 4
13: CATEND

14: END
```

- `BOX` in line 1 is generated
- `SPHERE` in line 3 is skipped (since there is an active signal)
- `CATBEG/CATEND` in lines 4–6 are skipped (since they do not match `99`)
- `SPHERE` in line 7 is skipped
- `BOX` in line 9 is generated
- `CATBEG/CATEND` in lines 11–13 are skipped (since the signal was cancelled when it was caught in line 8)

Special Note on Programming Blocks

- Programming Blocks are delineated by
  - `PATBEG` and `PATEND`
  - `IFTHEN`, `ELSEIF`, `ELSE`, and `ENDIF`
  - `SOLBEG` and `SOLEND`
  - `CATBEG` and `CATEND`
- Any programming Block can be nested fully within any other programming Block (up to 20 levels deep)

Reading Help File (1)

```
STORE      $name index=0 keep=0
           use:    stores Group on top of Stack
           pops:   any
           pushes: -
           notes:  Sketch may not be open
                   Solver may not be open
                   $name is used directly (without evaluation)
                   previous Group in name/index is overwritten
                   if $name=.   then Body is popped off stack
                                     but not actually stored
                   if $name=..  then pop Bodys off stack back
                                     to the Mark
                   if $name=... then the stack is cleared
                   if keep==1, the Group is not popped off stack
                   cannot be followed by ATTRIBUTE or CSYSTEM
                   signals that may be thrown/caught:
                      $insufficient_bodys_on_stack
```

Reading Help File (2)

- If argument starts with dollar-sign ($), then the argument is assumed to be string, and the user does not need to prepend the argument with a dollar-sign ($)
  - if an expression is given that should be evaluated (to a string value), prepend the argument with an exclamation point (!), as in:

```
SET    i 10
STORE  !$ThisIsBody+i+$.
```

    stores the Body in a location named `ThisIsBody10.`
- For arguments that are listed with an equal-sign (=), the value after the equal sign is the default value

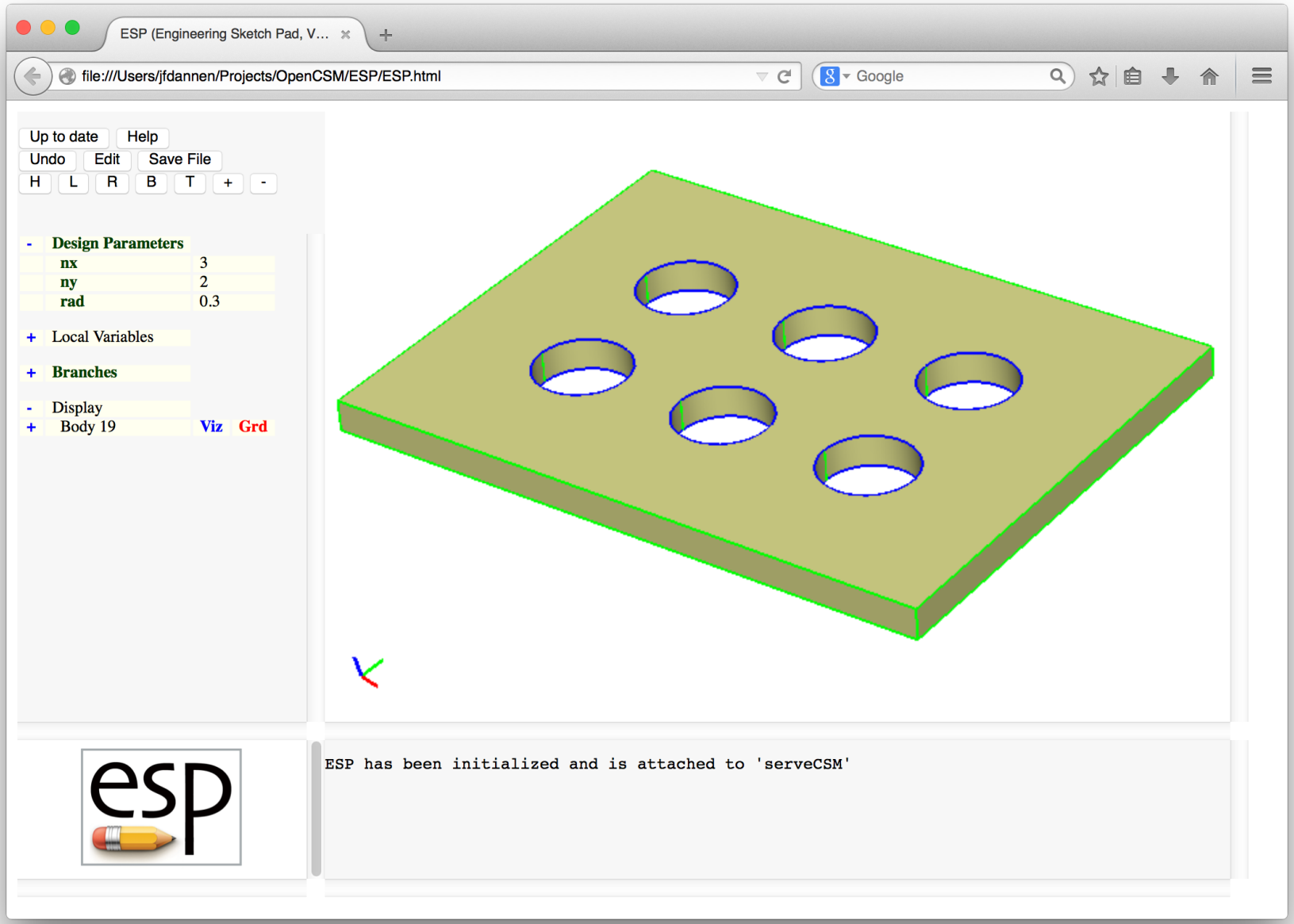
Other Sources of Help and/or Examples

- `ESP-help` $\Longrightarrow$ "Valid CSM statements"
  - Examples of the use of each command can be obtained by pressing the small picture at the end of the associated help section
- `ESP-help` $\Longrightarrow$ "User-defined Primitives/Functions shipped with OpenCSM"
  - Examples of the use of each UDP/UDF can be obtained by pressing the small picture at the end of the associated help section
- `$ESP_ROOT/data/basic`
  - This directory contains many of the files that are used for nightly testing. You can search through this directory for `.csm` files that use the command in which you are interested.

Homework Exercises

- Rectangular plate with holes
- Round plate with holes
- Determine if two Bodys overlap
- Files in `$ESP_ROOT/training/ESP/exercises/session03` will get you started

Rectangular Plate with Holes (1)



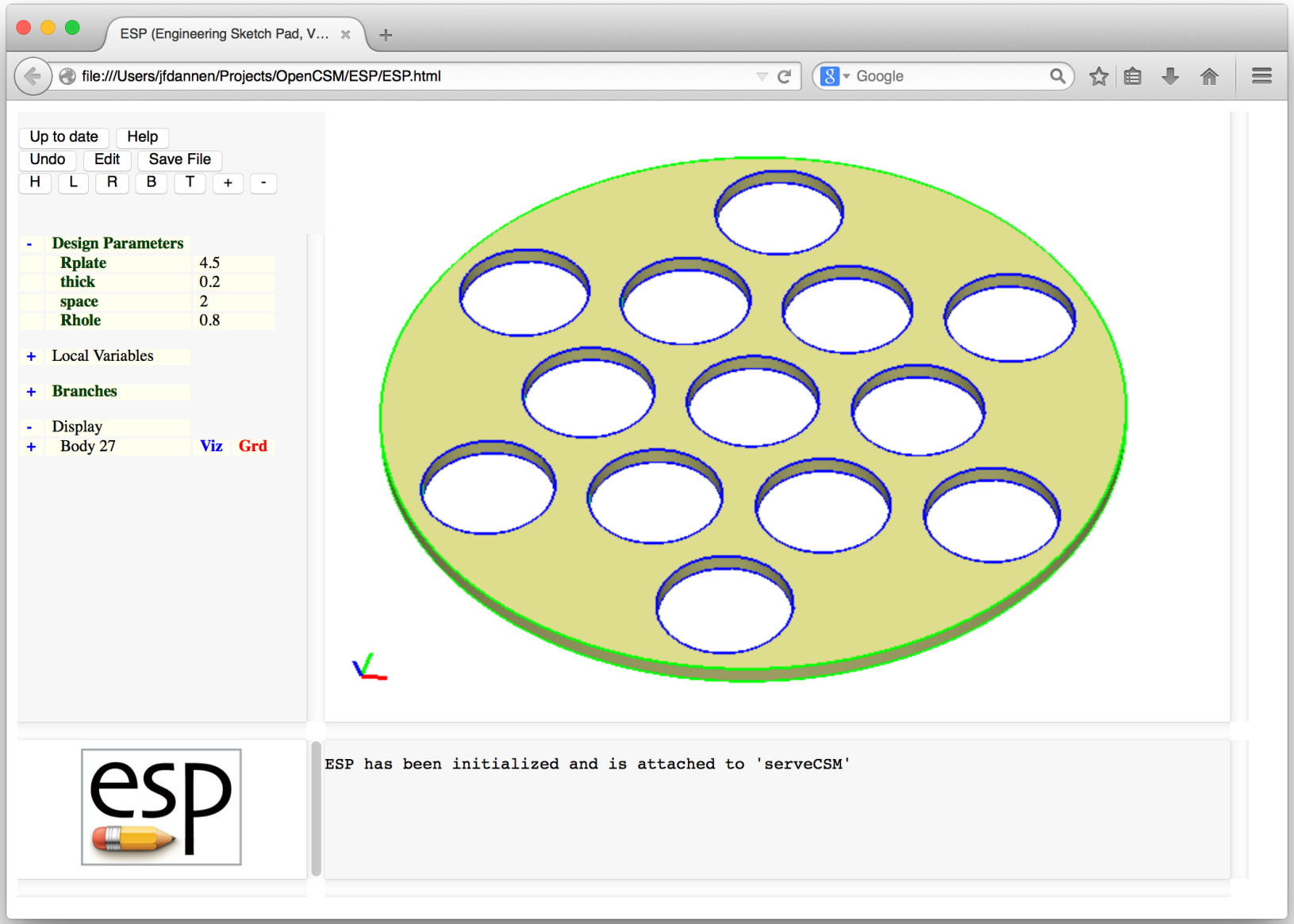
Rectangular Plate with Holes (2)

nx	number of holes in X-direction	3.00
ny	number of holes in Y-direction	2.00
rad	radius of each hole	0.30
space	distance between hole centers	1.00
edge	distance between holes and plate edge	0.50
thick	plate thickness	1.00

Rectangular Plate with Holes (3)

- Which parameter(s) should be a DESPMTR and which should be a CFGPMTR?
- What if you make the radius of the hole too big?
- What happens if you make the plate thickness zero?

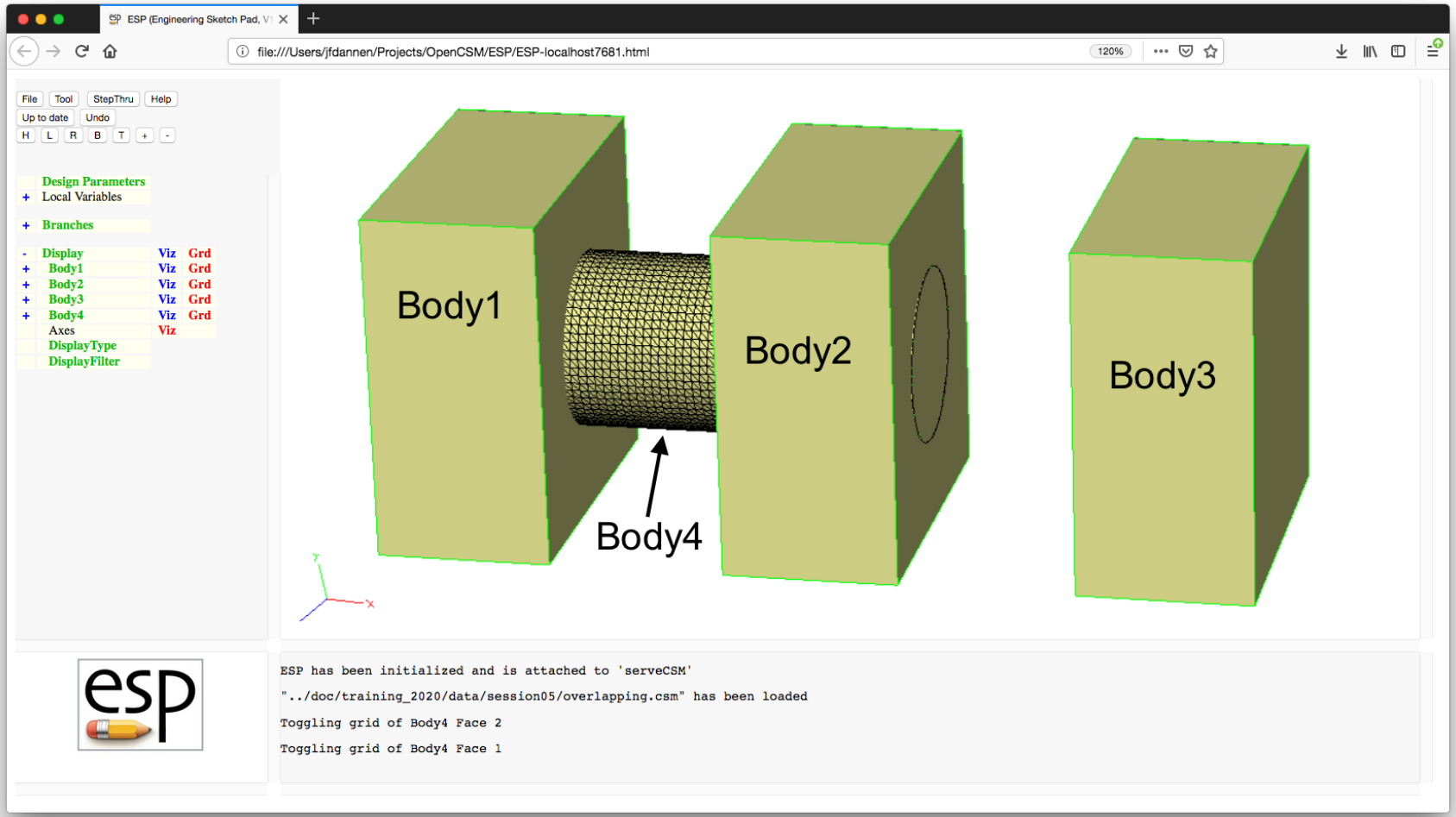
Round Plate with Holes (1)



Round Plate with Holes (2)

Rplate	radius of plate	4.50
thick	thickness of plate	0.20
space	distance between hole centers	2.00
Rhole	radius of holes	0.80
	number of holes selected automatically	

Overlapping Bodys (1)



Overlapping Bodys (2)

- Write `.csm` file to:
  - set `overlap1` to 1 if Bodys 1 and 4 overlap, otherwise set it to 0
  - set `overlap2` to 1 if Bodys 2 and 4 overlap, otherwise set it to 0
  - set `overlap3` to 1 if Bodys 3 and 4 overlap, otherwise set it to 0
- Try to use a pattern to do this compactly