Engineering Sketch Pad (ESP) Training
Session 4: CSM Scripts

John F. Dannenhoffer, III

Syracuse University

Bob Haimes

Massachusetts Institute of Technology

Revised for v1.09

Overview

- Format of the `.csm` file
- Special characters
- Valid CSM statements
- Number rules
- Parameter rules
- Expression rules
- Attribute and Csystem rules
- Patterns, If/then, and Throw/catch
- Hands-on exercise
  - rectangular plate
  - round plate

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.

Any CSM statement can be used except the INTERFACE statement.

Blocks of statements must be properly nested. The Blocks are bounded
by PATBEG/PATEND, IFTHEN/ELSEIF/ELSE/ENDIF, SOLBEG/SOLEND,
and CATBEG/CATEND.

Format of the .csm file (2)

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.

All arguments must not contain any spaces or must be enclosed
in a pair of double quotes (for example, "a + b").

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 LIFO 'Stack' that
can contain Bodys 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

```
#           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
( )         groups expressions and function arguments
[ ]         specifies subscripts in form [row,column] or [index]
+ - * / ^   arithmetic operators
$           as first character, forces argument not to be evaluated
               (used internally)
@           as first character, introduces @-parameters (see below)
!           evaluate before setting attribute

~ % & = '   not used
{ } < > ?   not used
```


Number Rules

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

Parameter Rules (1)

```
Valid names:
    start with a letter, colon (:), or at-sign (@)
    contains letters, digits, at-signs (@), underscores (_), and colons (:)
    contains fewer than 32 characters
    names that start with an at-sign cannot be set by a CONPMTR, DESPMTR,
       or SET statement
    if a name has a dot-suffix, a property of the name (and not its value) is
       returned
       x.nrow   number of rows     in x
       x.ncol   number of columns  in x
       x.size   number of elements in x (=x.nrow*x.ncol)
       x.sum    sum of elements    in x
       x.norm   norm of elements   in x (=sqrt(x[1]^2+x[2]^2+...))
       x.min    minimum value      in x
       x.max    maximum value      in x

Array names:
    basic format is: name[irow,icol] or name[ielem]
    name must follow rules above
    irow, icol, and ielem must be valid expressions
    irow, icol, and ielem start counting at 1
    values are stored across rows ([1,1], [1,2], ..., [2,1], ...)
```


Parameter Rules (2)

```
Types:
    CONSTANT
       declared and defined by a CONPMTR statement
       must be a scalar
        is only available at the .csm file level
        can be set  outside ocsmBuild by a call to ocsmSetValu
        can be read outside ocsmBuild by a call to ocsmGetValu
    EXTERNAL
        if a scalar, declared and defined by a DESPMTR statement
        if an array, declared by a DIMENSION statement (with despmtr=1)
                     values defined by one or more DESPMTR statements
        each value can only be defined in one DESPMTR statement
        can have an optional lower bound
        can have an optional upper bound
        is only available at the .csm file level
        can be set  outside ocsmBuild by a call to ocsmSetValu
        can be read outside ocsmBuild by a call to ocsmGetValu
    INTERNAL
        if a scalar, declared and defined by a SET statement
        if an array, declared by a DIMENSION statement (with despmtr=0)
                     values defined by one or more SET statements
        values can be overwritten by subsequent SET statements
```

Parameter Rules (3)

@-parameters depend on the last SELECT statement.
each time a new Body is added to the Stack, 'SELECT body' is implicitly called
depending on last SELECT statement, the values of the @-parameters are given by:

	body	face	edge	node	<- last SELECT
@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

Parameter Rules (4)

@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	

Parameter Rules (5)

```
@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

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


Parameter Rules (6)

```
Scope:
    CONSTANT parameters are available everywhere
    EXTERNAL parameters are only usable within the .csm file
    INTERNAL within a .csm file
                created by a DIMENSION or SET statement
                values are usable only within the .csm file
             within a .udc file
                created by an INTERFACE of SET statament
                values are usable only with the current .udc file
```

Expression Rules (1)

```
Valid operators (in order of precedence):
    ( )            parentheses, inner-most evaluated first
    func(a,b)      function arguments, then function itself
    ^              exponentiation      (evaluated left to right)
    * /            multiply and divide (evaluated left to right)
    + -            add and subtract    (evaluated left to right)
```


Expression Rules (2)

```
Valid function calls:
    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 -> 3, -3.5 -> -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
    mod(a,b)                modulus(a/b), with same sign as a and b>=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
```

Expression Rules (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)`
`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)`

Expression Rules (4)

`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`
`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 degrees) produces derivative=0`
`turnang(xa,ya,dab,xb,yb,... dbc,xc,yc)`	`turnnig 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<=x<=180`

Expression Rules (5)

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


Attribute Rules (1)

- Attributes are defined for any Branch that produces a Body
- Attributes are defined by an `attribute` statement
- Attribute names must not start with a period (which is reserved for `EGADS`) or an underscore (which is reserved for `OpenCSM`)
- If the first character of the value is a dollar-sign, then the Attribute will contain a character string
- Otherwise the Attribute will contain one or more real (double) values
  - if the value is the name of a multi-valued Parameter, then the Attribute will be multi-valued
  - if the value is a semi-colon-separated list of expressions, then the Attribute will be multi-valued
  - otherwise the Attribute will be a single real (double)

Attribute Rules (2)

```
EGADS attributes assigned to Bodys:
    body        Body index (bias-1)
    brch        Branch index (bias-1)
    <any>       all global attributes
    <any>       all attributes associated with Branch that created Body
    <any>       all attributes associated with "select $body" statement
```


Attribute Rules (3)

```
EGADS attributes assigned to Faces:
    body        non-unique 2-tuple associated with first Face creation
        [0]     Body index in which Face first existed (bias-1)
        [1]     face-order associated with creation (see above)
    brch        non-unique even-numbered list associated with Branches
                   that are active when the Face is created (most
                   recent Branch is listed first)
        [2*i  ] Branch index (bias-1)
        [2*i+1] (see below)

                Branches that contribute to brch attribute are
                   primitive  (for which brch[2*i+1] is face-order)
                   udprim.udc (for which brch[2*i+1] is 1)
                   grown      (for which brch[2*i+1] is face-order)
                   applied    (for which brch[2*i+1] is face-order)
                   sketch     (for which brch[2*i+1] is Sketch primitive if
                               making WIRE)
                   patbeg     (for which brch[2*i+1] is pattern index)
                   recall     (for which brch[2*i+1] is 1)
                   restore    (for which brch[2*i+1] is Body number stored)
```

Attribute Rules (4)

```
faceID      unique 3-tuple that is assigned automatically
      [0]   body[0]
      [1]   body[1]
      [2]   sequence number

            if multiple Faces have same faceID[0] and faceID[1],
               then the sequence number is defined based upon the
               first rule that applies:
               * Face with smaller xcg  has lower sequence number
               * Face with smaller ycg  has lower sequence number
               * Face with smaller zcg  has lower sequence number
               * Face with smaller area has lower sequence number
<any>       all attributes associated with Branch that first created Face
<any>       all attributes associated with "select $face" statement
```


Attribute Rules (5)

EGADS attributes assigned to Edges:

```
body        non-unique 2-tuple associated with first Edge creation
    [0]     Body index in which Edge first existed (bias-1)
    [1]     100 * min(body[1][ileft],body[1][irite])
                + max(body[1][ileft],body[1][irite])
            (or -3 if non-manifold)
edgeID      unique 5-tuple that is assigned automatically
      [0]   faceID[0] of Face 1 (or 0 if non-manifold)
      [1]   faceID[1] of Face 1 (or 0 if non-manifold)
      [2]   faceID[0] of Face 2 (or 0 if non-manifold)
      [3]   faceID[1] of Face 2 (or 0 if non-manifold)
      [4]   sequence number

            edgeID[0]/[1] swapped with edge[2]/[3]
               100*edgeID[0]+edgeID[1] > 100*edgeID[2]+edgeID[3]
            if multiple Edges have same edgeID[0], edgeID[1],
               edgeID[2], and edgeID[3], then the sequence number
               is defined based upon the first rule that applies:
               * Edge with smaller xcg    has lower sequence number
               * Edge with smaller ycg    has lower sequence number
               * Edge with smaller zcg    has lower sequence number
```

Attribute Rules (6)

```
EGADS attributes assigned to Nodes:
    nodeID      not assigned at this time
    <any>       all attributes associated with "select $node" statement
```


Csystem rules (1)

- Csystems (coordinate systems) are generated by the `csystem` statement and are applied to the Body on the top of the Stack
- Csystems are treated in many ways like Attributes
  - Csystem names must not be the same as a Attribute name
  - Csystems are found in ESP in same place as Attributes
- Csystems are transformed along with any transformations that are applied to their Body

Csystem rules (2)

- Format of the csystem statement is:
  - If argument to csystem contains 9 entries:

```
{x0, y0, z0, dx1, dy1, dz1, dx2, dy2, dz3}
origin is at (x0,y0,q0)
dirn1  is in (dx1, dy1,dz1) direction
dirn2  is in (dx2,dy2,dz2)  direction
```

  - If argument to csystem contains 5 entries and first is positive:

```
{+iface, ubar0, vbar0, du2, dv2}
origin is at normalized (ubar0,vbar0) in iface
dirn1  is normal to Face
dirn2  is in (du2,dv2) direction
```


Csystem rules (3)

- Format of the csystem statement is:
  - If argument to csystem contains 5 entries and first is negative:

```
{-iedge, tbar, dx2, dy2, dz2}
origin is at normalized (tbar) in iedge
dirn1  is tangent to Edge
dirn2  is in (dx2,dy2,dz2) direction
```

  - If argument to csystem contains 7 entries:

```
{inode, dx1, dy1, dz1, dx2, dy2, dz2}
origin is at Node inode
dirn1  is in (dx1,dy1,dz1) direction
dirn2  is in (dx1,dy2,dz2) direction
```

Patterns

- 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
- Example pattern (indentation optional):

```
PATBEG     i    7
   SET     j    i-1
   BOX     j    0  0  1  1  1
   ROTATEX j*10 0  0
PATEND
```


If/then (1)

- If/then constructs are used to make 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` options have arguments
  - `val1` — an expression
  - `op1` — can be `lt`, `le`, `eq`, `ge`, `gt`, or `ne`
  - `val2` — an expression
  - `op2` — can be `or` or `and` (defaults to `and`)
  - `val3` — an expression (defaults to `0`)
  - `op3` — can be `lt`, `le`, `eq`, `ge`, `gt`, or `ne` (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
```


Throw/catch

- Throw/catch constructs are used to generate and react to signals (errors)
- Signals can be generated by
  - executing a `throw` command
  - a run-time error encountered elsewhere (see "help" for more info)
- When a signal is generated, all Braches 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

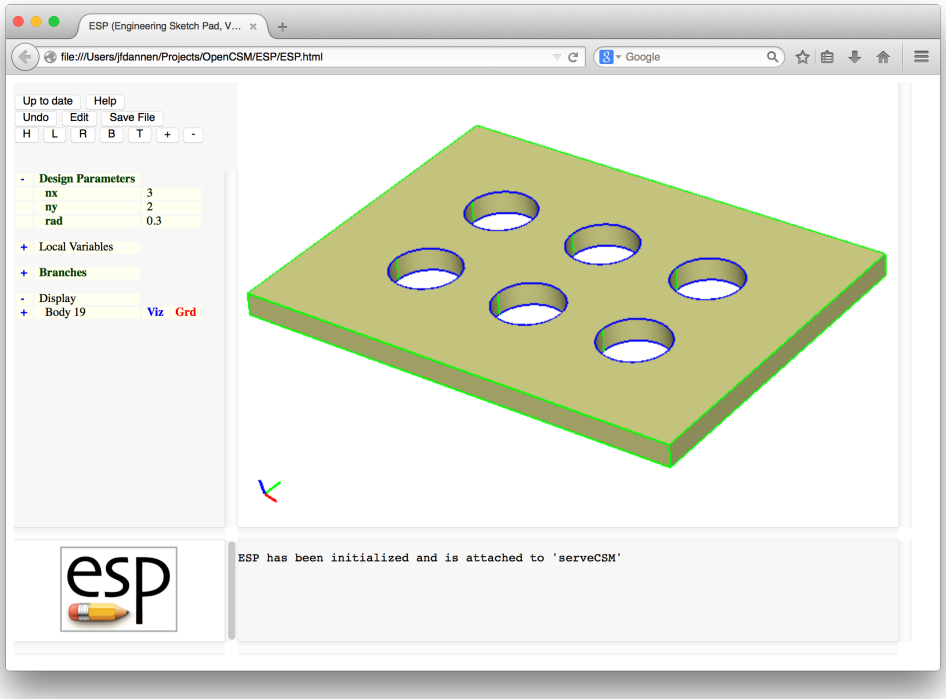
Special Note on Programming Blocks

- Programming Blocks are delineated by
  - `patbeg` and `patend`
  - `ifthen`, `elseif`, `else`, and `endif`
  - `catbeg` and `catend`
- Any programming Block can be nested fully within any other programming Block (up to 10 levels deep)

Hands-on Exercises

- Rectangular pattern
  - holes in rows and columns
- Round plate with holes
  - holes in regular 2D pattern
  - requires additional pattern to determine if candidate hole is "within" the plate

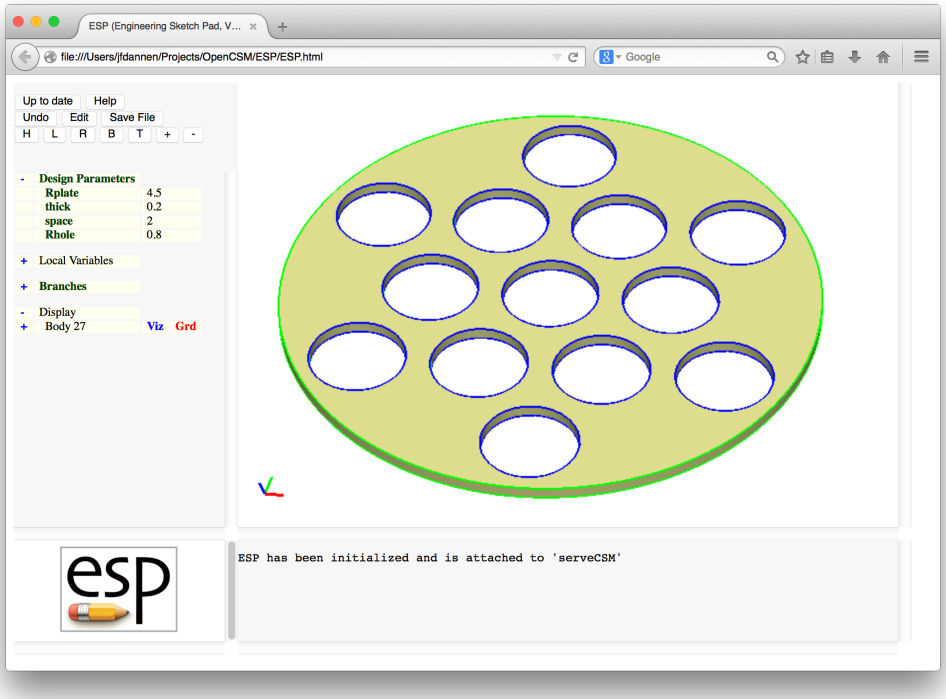
Rectangular Pattern (1)



Rectangular Pattern (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
	distance between hole centers	1.00

Round Plate with Holes (1)



Round Plate with Holes (2)

Rplate	radius or 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	

Muddy Cards

- Are there any items that still confuse you
- Is it clear how to make decisions using a pattern?
- Is there anything else that is unclear