

Engineering Sketch Pad (ESP) Training

Session 4: CSM Scripts

John F. Dannenhoffer, III

Syracuse University

Bob Haimes

Massachusetts Institute of Technology

Revised for v1.13





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.



Format of the .csm file (2)

- Blocks of statements must be properly nested. The Blocks are bounded by
 - PATBEG/PATEND
 - IFTHEN/ELSEIF/ELSE/ENDIF
 - 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.
- 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 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 (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)

<code>()</code>	groups expressions and function arguments
<code>[]</code>	specifies subscripts in form <code>[row,column]</code> or <code>[index]</code>
<code>+ - * / ^</code>	arithmetic operators
<code>\$</code>	as first character, forces argument not to be evaluated (used internally)
<code>@</code>	as first character, introduces @-parameters (see below)
<code>!</code>	if first character of implicit string, ignore \$! and treat as an expression
<code>~ % & = ' ,</code>	not used
<code>{ } < > ?</code>	not used



Number Rules

- 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 (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.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`



Parameter Rules (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 (Types — 1)

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 ocsmssetvalu

- can be read outside ocsmbuild by a call to ocsmsgetvalu

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 ocsmssetvalu

- can be read outside ocsmbuild by a call to ocsmsgetvalu

...



Parameter Rules (Types — 2)

Types:

...

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

- are created by an INTERFACE statement in a .udc file

- see scope rules (below)



Parameter Rules (At-parameters — 1)

@-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
@seltype	-1	2	1	0	selection type (0=node,1=edge,2=face)
@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



Parameter Rules (At-parameters — 2)

@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 (At-parameters — 3)

@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 (scope)

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 statement

- values are usable only with the current .udc file

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)



Expression Rules (Functions — 1)

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



Expression Rules (Functions — 2)

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



Expression Rules (Functions — 3)

<code>sin(x)</code>	sine of x	(in radians)
<code>sind(x)</code>	sine of x	(in degrees)
<code>asin(x)</code>	arc-sine of x	(in radians)
<code>asind(x)</code>	arc-sine of x	(in degrees)
<code>cos(x)</code>	cosine of x	(in radians)
<code>cosd(x)</code>	cosine of x	(in degrees)
<code>acos(x)</code>	arc-cosine of x	(in radians)
<code>acosd(x)</code>	arc-cosine of x	(in degrees)



Expression Rules (Functions — 4)

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



Expression Rules (Functions — 5)

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



Expression Rules (Functions — 6)

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



Expression Rules (Functions — 7)

`ifzero(test,ifTrue,ifFalse)`

if $\text{test} = 0$, return `ifTrue`,
else return `ifFalse`

`ifpos(test,ifTrue,ifFalse)`

if $\text{test} > 0$, return `ifTrue`,
else return `ifFalse`

`ifneg(test,ifTrue,ifFalse)`

if $\text{test} < 0$, return `ifTrue`,
else return `ifFalse`

`ifnan(test,ifTrue,ifFalse)`

if test is NaN, return `ifTrue`,
else return `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:

<code>_body</code>	Body index (bias-1)
<code>_brch</code>	Branch index (bias-1)
<code>_tParams</code>	tessellation parameters
<code>_csys_*</code>	arguments when CSYSTEM was defined
<code><any></code>	all global attributes
<code><any></code>	all attributes associated with Branch that created Body
<code><any></code>	all attributes associated with "select \$body" statement



Attribute Rules (3)

EGADS attributes assigned to Faces:

<code>_body</code>	non-unique 2-tuple associated with first Face creation
<code>[0]</code>	Body index in which Face first existed (bias-1)
<code>[1]</code>	face-order associated with creation (see above)
<code>_brch</code>	non-unique even-numbered list associated with Branches that are active when the Face is created (most recent Branch is listed first)
<code>[2*i]</code>	Branch index (bias-1)
<code>[2*i+1]</code>	(see below)

Branches that contribute to `brch` attribute are

primitive	(for which <code>brch[2*i+1]</code> is face-order)
udprim.udc	(for which <code>brch[2*i+1]</code> is 1)
grown	(for which <code>brch[2*i+1]</code> is face-order)
applied	(for which <code>brch[2*i+1]</code> is face-order)
sketch	(for which <code>brch[2*i+1]</code> is Sketch primitive if making WIRE)
patbeg	(for which <code>brch[2*i+1]</code> is pattern index)
recall	(for which <code>brch[2*i+1]</code> is 1)
restore	(for which <code>brch[2*i+1]</code> 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

`_hist` list of Bodys that contained this Face (oldest to newest)

`<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:

<code>_body</code>	non-unique 2-tuple associated with first Edge creation
<code>[0]</code>	Body index in which Edge first existed (bias-1)
<code>[1]</code>	$100 * \min(\text{body}[1][\text{ileft}], \text{body}[1][\text{irite}])$ $+ \max(\text{body}[1][\text{ileft}], \text{body}[1][\text{irite}])$ (or -3 if non-manifold)
<code>_edgeID</code>	unique 5-tuple that is assigned automatically
<code>[0]</code>	<code>_faceID[0]</code> of Face 1 (or 0 if non-manifold)
<code>[1]</code>	<code>_faceID[1]</code> of Face 1 (or 0 if non-manifold)
<code>[2]</code>	<code>_faceID[0]</code> of Face 2 (or 0 if non-manifold)
<code>[3]</code>	<code>_faceID[1]</code> of Face 2 (or 0 if non-manifold)
<code>[4]</code>	sequence number

...



Attribute Rules (6)

EGADS attributes assigned to Edges:

...

```
_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
  * Edge with smaller length has lower sequence number
```

_nface

number of incident Faces

<any>

all attributes associated with "select \$edge" statement



Attribute Rules (7)

EGADS attributes assigned to Nodes:

<code>_nodeID</code>	unique integer
<code>_nedge</code>	number of incident Edges
<code><any></code>	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

- 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 part of (dx2,dy2,dz2) that is
orthogonal to dirn1
 - 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 part of (dx1,dy2,dz2) that is
orthogonal to dirn1

- 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 statements have arguments
 - val1 — an expression
 - op1 — can be lt, le, eq, ge, gt, or ne
 - val2 — an expression
 - op2 — can be or, xor, 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 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



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)

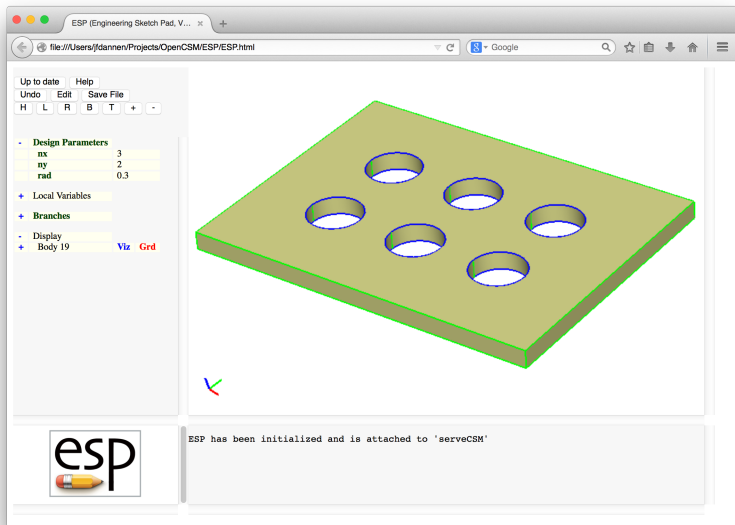


Hands-on Exercises

- 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)



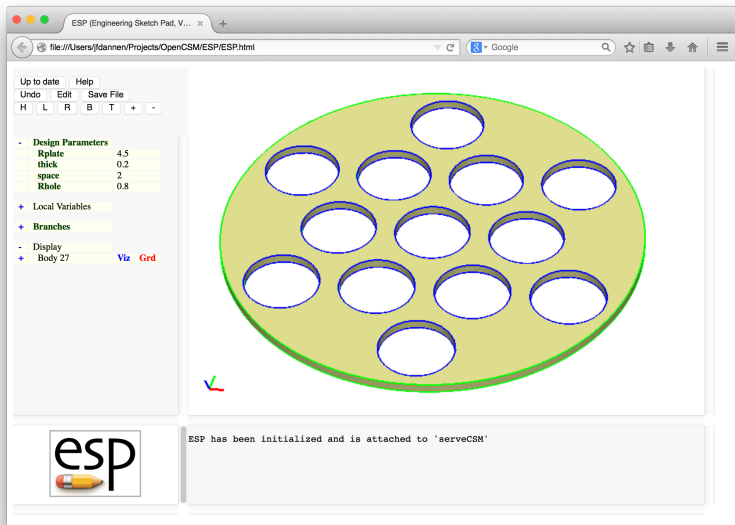


Rectangular Pattern (2)

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 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	



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