

Engineering Sketch Pad (ESP) Training

Session 4: Expressions and Patterns

John F. Dannenhoffer, III

Syracuse University

Bob Haimes

Massachusetts Institute of Technology

Revised for v1.08





Overview

- Format of the .csm file
- Special characters
- Valid CSM statements
- Number rules
- Parameter rules
- Expression rules
- Attribute rules
- Patterns
- 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.

Any CSM statement can be used except the interface statement.

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.



Format of the .csm file (2)

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 Bodies 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



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 `despmtr` or `set` statement
- if a name has a dot-suffix, a property of the name (and not its value) is returned

<code>x.nrow</code>	number of rows	in <code>x</code>
<code>x.ncol</code>	number of columns	in <code>x</code>
<code>x.size</code>	number of elements	in <code>x</code> ($=x.nrow \times x.ncol$)
<code>x.sum</code>	sum of elements	in <code>x</code>
<code>x.norm</code>	norm of elements	in <code>x</code> ($=\sqrt{x[1]^2 + x[2]^2 + \dots}$)
<code>x.min</code>	minimum value	in <code>x</code>
<code>x.max</code>	maximum value	in <code>x</code>

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:

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

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)

SOLVER

- not implemented yet



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

@ibody1 -1 x x -1 first element of 'Body' attribute in @ibody

@ibody2 -1 x x -1 second element of 'Body' attribute in @ibody



Parameter Rules (4)

@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	

Scope:

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

SOLVER parameters are only accessible between SOLBEG and SOLEND statements



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:

<code>pi(x)</code>	<code>3.14159...*x</code>
<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 -> 3, -3.5 -> -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
<code>mod(a,b)</code>	modulus(a/b), with same sign as a and b>=0
<code>sign(test)</code>	returns -1, 0, or +1 produces derivative=0
<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 (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)
<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 (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 $-180 \leq x \leq 180$



Expression Rules (5)

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



Attribute Rules (1)

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 (2)

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>i</i>]	Branch index (bias-1)
[2 <i>i</i> +1]	(see below)

Branches that contribute to brch attribute are

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



Attribute Rules (3)

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 (4)

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] $100 * \min(\text{body}[1][\text{ileft}], \text{body}[1][\text{irite}])$
 $+ \max(\text{body}[1][\text{ileft}], \text{body}[1][\text{irite}])$
 (or -3 if non-manifold)

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]

$100 * \text{edgeID}[0] + \text{edgeID}[1] > 100 * \text{edgeID}[2] + \text{edgeID}[3]$

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 (5)

EGADS attributes assigned to Nodes:

nodeID not assigned at this time

<any> all attributes associated with "select \$node" statement

- 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 patend statement
 - Patterns can be used to make decisions
 - `set foo sqrt(4)`
 - `patbeg icopy ifzero(foo-2,1,0)`
 - ...
 - `patend`
- is equivalent to
- `foo = sqrt(4);`
 - `if (foo == 2) { ... }`

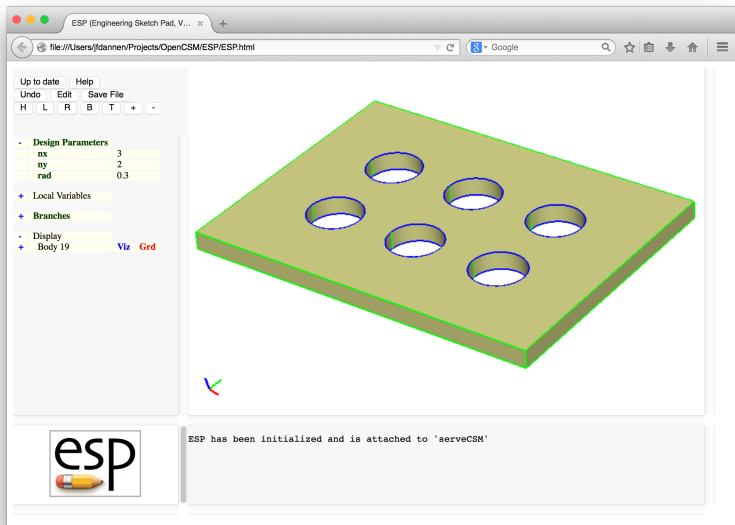


Hands-on Exercises

- 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)



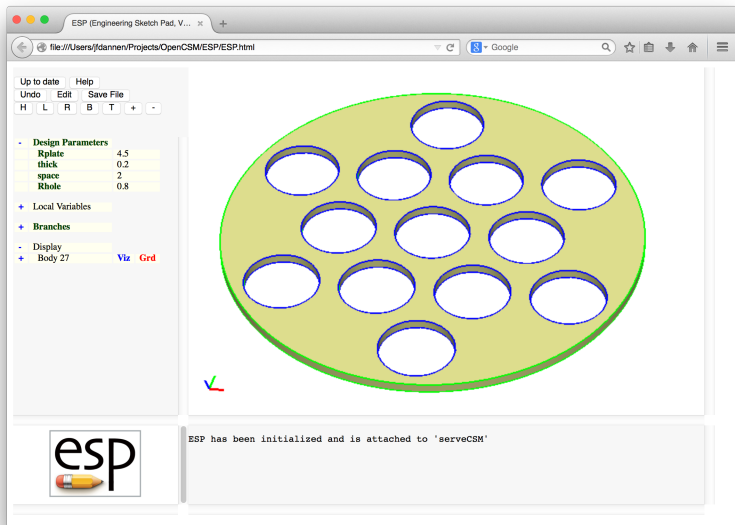


Rectangular Pattern (2)

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