

# Engineering Sketch Pad (ESP)



## New User Training Session 03 CSM Language

**John F. Dannenhoffer, III**

[john@geocentrictech.com](mailto:john@geocentrictech.com)

Geocentric Technologies LLC

updated for v1.29+

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

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

- 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

- 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.

- Blocks of statements must be properly nested. The Blocks are bounded by
  - PATBEG/PATEND
  - IFTHEN/ELSEIF/ELSE/ENDIF
  - SKBEG/SKEND
  - SOLBEG/SOLEND
  - OPTBEG/OPTEND
  - 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").

- 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.

|         |                                                                            |
|---------|----------------------------------------------------------------------------|
| #       | introduces comment                                                         |
| "       | ignore spaces until following "                                            |
| \       | ignore this and following characters and<br>concatenate next line          |
| <space> | separates arguments in .csm file (except<br>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),<br>introduces dot-suffixes (in names) |
| ,       | separates function arguments and row/column<br>in subscripts               |
| ;       | multi-value item separator                                                 |

|           |                                                                                                                                 |
|-----------|---------------------------------------------------------------------------------------------------------------------------------|
| ( )       | 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<br>terminated by end-of-line or un-escaped plus,<br>comma, or close-parenthesis |
| @         | as first character, introduces @-parameters                                                                                     |
| ,         | used to escape comma, plus, or close-parenthesis<br>within strings                                                              |
| !         | if first character of implicit string, ignore<br>\$! and treat as an expression                                                 |
|           | cannot be used (reserved for OpenCSM internals)                                                                                 |
| &         | cannot be used (reserved for OpenCSM internals)                                                                                 |

- 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 (;)

- 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 DIMENSIONed
  - 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`

- 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 `DIMENSIONED`
  - 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

- 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

- 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 **DIMENSIONED**
  - 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**)

|                                                                                                    | 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 <code>ocsmLoad</code> or <code>ocsmLoadDict</code>                                  | Y       | Y       | Y       | N       | N        |
| Can be set via <code>ocsmSetValu(D)</code>                                                         | Y       | Y       | N       | N       | N        |
| Defined and set during <code>ocsmBuild</code>                                                      | N       | N       | N       | Y       | Y        |
| Can be read via <code>ocsmGetValu(S)</code>                                                        | Y       | Y       | Y       | Y       | Y*       |
| Can find associated sensitivity                                                                    | Y       | N       | N       | N       | N        |
| Y*=Parameter index may be different for different builds<br>scopes: T=top-level, G=global, L=local |         |         |         |         |          |

- 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

- 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 (:)

- The **PREFIX** command specifies a prefix that is prepended to any Parameter name that starts with a colon (:)
- The prefix remains in effect until another **PREFIX** command is encountered or **PREFIX .** is used to clear the prefix
- For example:

```
PREFIX the:fuselage  
SET      :width 3
```

- is equivalent to:

```
SET      the:fuselage:width 3
```

- The **prefix()** function can be used to get the current prefix

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

|                     |                                                    |
|---------------------|----------------------------------------------------|
| <code>x.nrow</code> | number of rows in <code>x</code> (0 for string)    |
| <code>x.ncol</code> | number of columns in <code>x</code> (0 for string) |
| <code>x.size</code> | number of elements or characters in <code>x</code> |
| <code>x.sum</code>  | sum of elements in <code>x</code>                  |
| <code>x.norm</code> | RMS norm of elements in <code>x</code>             |
| <code>x.min</code>  | minimum value in <code>x</code>                    |
| <code>x.max</code>  | maximum value in <code>x</code>                    |

- Example:

```
DIMENSION  myvar  2 3
DESPMTR    myvar  "1; 2; 3;\n
              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
```

- 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]`, ...)

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

|          | 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,<br>2=SheetBody, 3=SolidBody |
| @nbors   | -1   | x    | -    | x    | number of incident Edges                            |
| @nbors   | -1   | -    | x    | -    | number of incident Faces                            |

|         |    |   |   |    |                                              |
|---------|----|---|---|----|----------------------------------------------|
| @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  |                                              |

|          |   |   |   |   |                                   |
|----------|---|---|---|---|-----------------------------------|
| @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

- 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)

- 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
```

|                       |                                                                                              |
|-----------------------|----------------------------------------------------------------------------------------------|
| <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 $x$ ( $3.5 \rightarrow 3$ , $-3.5 \rightarrow -3$ )<br>produces derivative=0 |
| <code>nint(x)</code>  | nearest integer to $x$<br>produces derivative=0                                              |
| <code>ceil(x)</code>  | smallest integer not less than $x$<br>produces derivative=0                                  |
| <code>floor(x)</code> | largest integer not greater than $x$<br>produces derivative=0                                |

|                         |                                                        |
|-------------------------|--------------------------------------------------------|
| <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                                   |
|                         | 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 $x$                                |

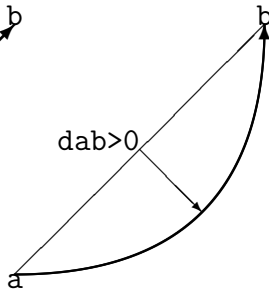
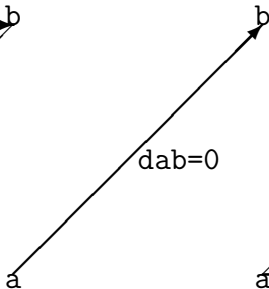
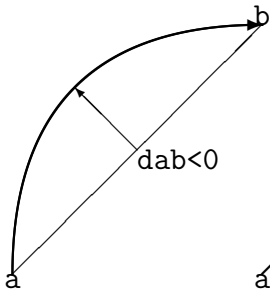
|                       |                   |              |
|-----------------------|-------------------|--------------|
| <code>sin(x)</code>   | sine of $x$       | (in radians) |
| <code>sind(x)</code>  | sine of $x$       | (in degrees) |
| <code>asin(x)</code>  | arc-sine of $x$   | (in radians) |
| <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: $\sqrt{x^2 + y^2 + z^2}$ |              |



# Circular Arc (Dip) Nomenclature

Arc going from point a to point b



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

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

|                                   |                                                                                        |
|-----------------------------------|----------------------------------------------------------------------------------------|
| <code>val2str(num,digits)</code>  | convert <code>num</code> to a string                                                   |
| <code>str2val(string)</code>      | convert <code>string</code> to a number                                                |
| <code>findstr(str1,str2)</code>   | finds location of <code>str2</code> in <code>str1</code><br>(bias-1) or 0 if not found |
| <code>slice(str,ibeg,iend)</code> | substring of <code>str</code> from <code>ibeg</code><br>to <code>iend</code> (bias-1)  |
| <code>ntoken(str)</code>          | get number of semicolon-separated<br>tokens                                            |
| <code>token(str,num)</code>       | get the n'th (bias-1) semicolon-<br>separated token from <code>str</code>              |
| <code>strlen(str)</code>          | number of characters in <code>str</code>                                               |

|                           |                                                                                                   |
|---------------------------|---------------------------------------------------------------------------------------------------|
| <code>filename()</code>   | returns <code>.csm</code> or <code>.udc</code> filename<br>(without directory but with extension) |
| <code>path(\$pwd)</code>  | returns present working directory                                                                 |
| <code>path(\$csm)</code>  | returns directory of current <code>.csm</code> file                                               |
| <code>path(\$root)</code> | returns <code>\$ESP_ROOT</code>                                                                   |
| <code>path(\$file)</code> | returns name of <code>.csm</code> file                                                            |
| <code>prefix()</code>     | returns current prefix or <code>--none--</code>                                                   |

|                                                   |                                                                                                                                                                                                                                                        |
|---------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>ifzero(test,ifTrue,ifFalse)</code>          | if <code>test = 0</code> , return <code>ifTrue</code> ,<br>else return <code>ifFalse</code>                                                                                                                                                            |
| <code>ifpos(test,ifTrue,ifFalse)</code>           | if <code>test &gt; 0</code> , return <code>ifTrue</code> ,<br>else return <code>ifFalse</code>                                                                                                                                                         |
| <code>ifneg(test,ifTrue,ifFalse)</code>           | if <code>test &lt; 0</code> , return <code>ifTrue</code> ,<br>else return <code>ifFalse</code>                                                                                                                                                         |
| <code>itmatch(str,pat,ifTrue,<br/>ifFalse)</code> | if <code>str</code> matches <code>pat</code> , return<br><code>ifTrue</code> , else return <code>ifFalse</code><br>wildcard in <code>pat</code> include:<br>? for any one character<br>'+' for one or more characters<br>* for zero or more characters |
| <code>ifnan(test,ifTrue,ifFalse)</code>           | if <code>test</code> is NaN, return <code>ifTrue</code> ,<br>else return <code>ifFalse</code>                                                                                                                                                          |

- 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
    - if only argument is given, breaks out if expression evaluates to a non-zero value
    - otherwise arguments are same as IFTHEN statement (see below)
  - patterns can be nested within other patterns

- 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 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)

- 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

```
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)

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

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    !$ThisIsBody+i+$.
```

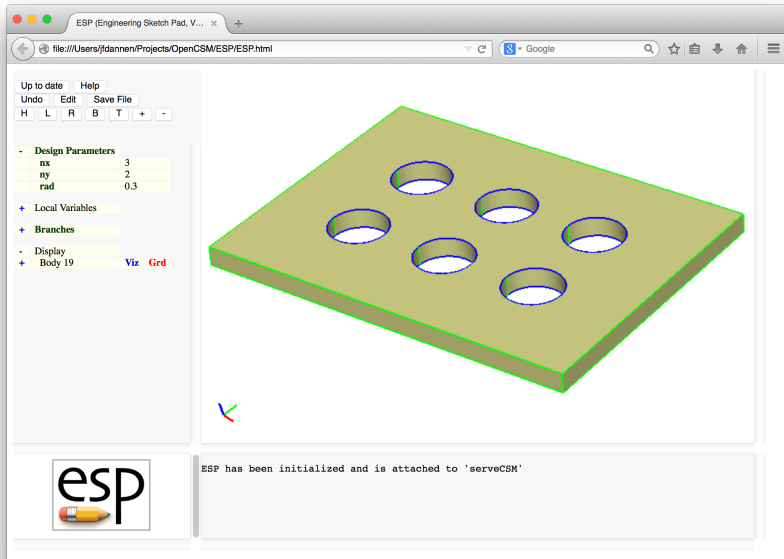
stores the Body in a location named `ThisIsBody10`.

- For arguments that are listed with an equal-sign (=), the value after the equal sign is the default value

- `ESP-help`  $\implies$  “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`  $\implies$  “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.

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

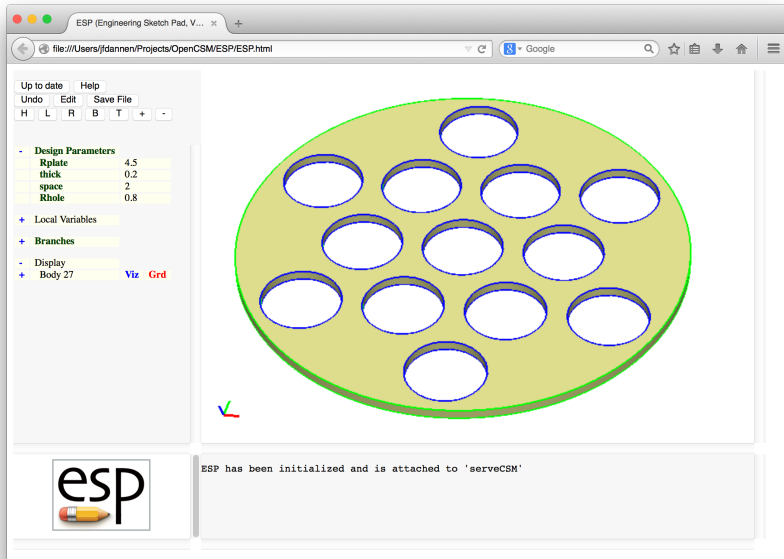
# Rectangular Plate with Holes (1)



|       |                                       |      |
|-------|---------------------------------------|------|
| 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 |

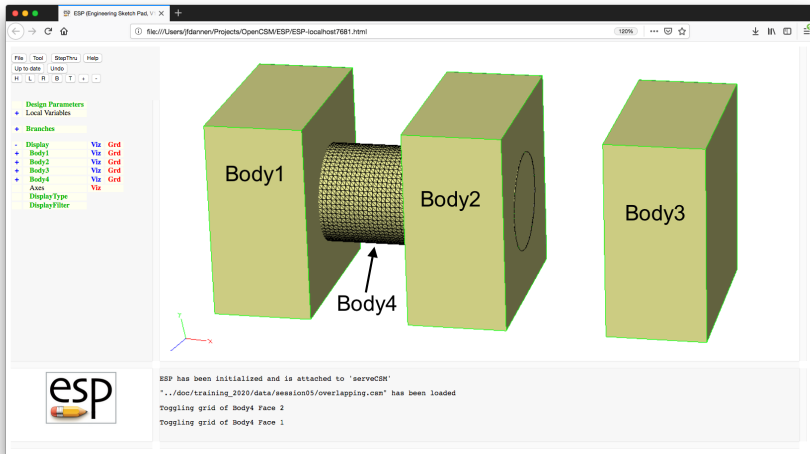
- Which parameter(s) should be a DESPMTR and which should be a CFGPMTR?
- What if you make the radius of the hole too big?
- What happens if you make the plate thickness zero?

# 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<br>automatically |      |



- 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