

Engineering Sketch Pad (ESP)



Training Session 1 ESP Overview & Getting Started

John F. Dannenhoffer, III

jfdannen@syr.edu
Syracuse University

Bob Haines

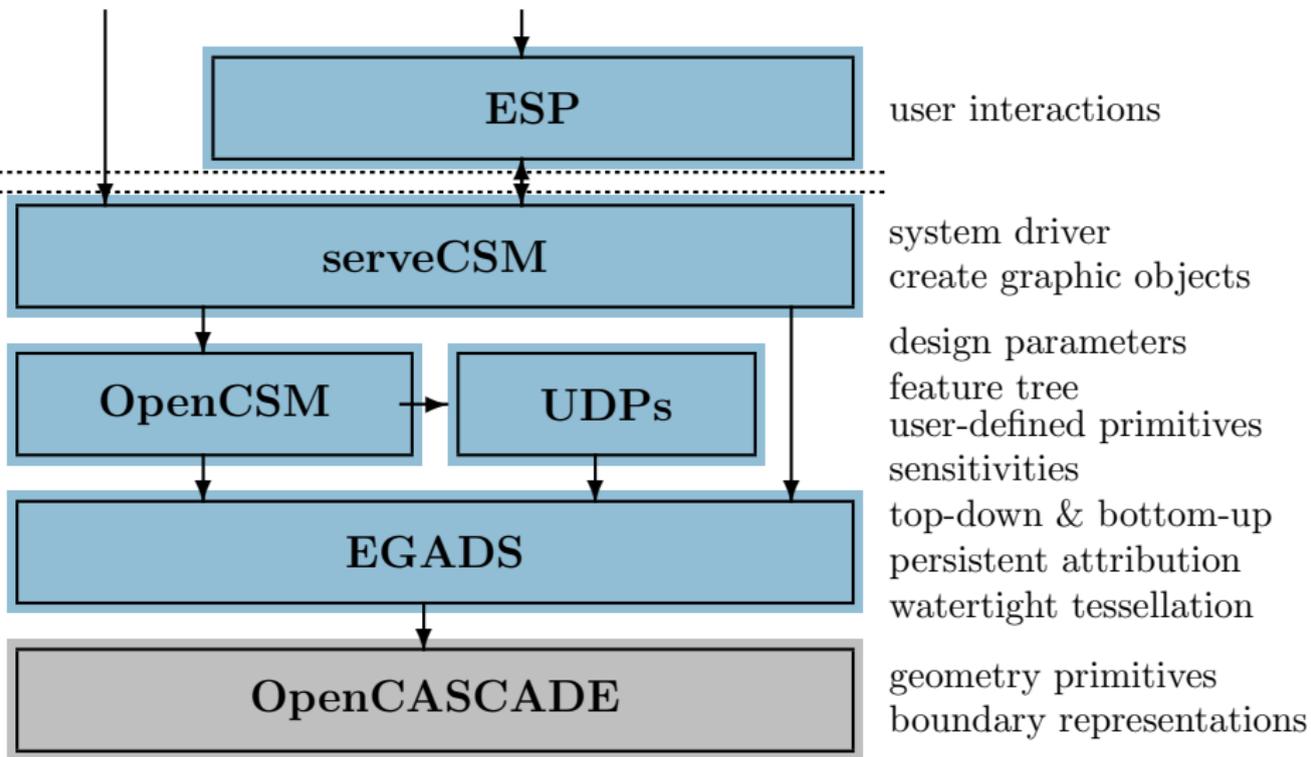
haines@mit.edu
Massachusetts Institute of Technology
updated for v1.18

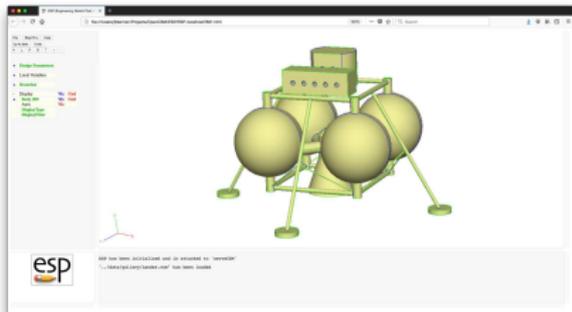
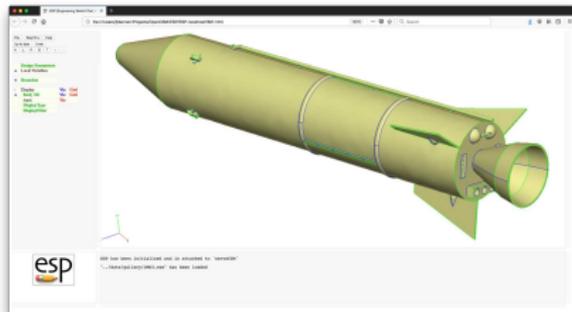
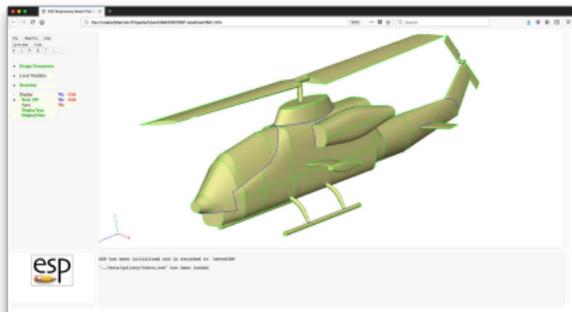
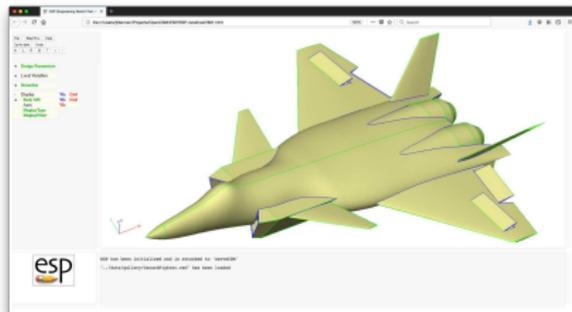
- ESP Overview
 - Background and Objectives
 - ESP Architecture
 - Distinguishing Features
- Starting ESP
- User Interface
 - Screen Layout
 - Image Manipulation
 - View Manipulation
- Getting Info
- StepThru Mode
- Journals & Exporting

- Over the past 40 years, there have been an increasingly-complex (complicated) series of “CAD” systems to support the geometry needs of the manufacturers of mechanical devices
 - CAD = “computer aided drafting”
 - CAD = “computer-aided drawing”
 - CAD = “computer-aided design”
 - CAD = “computer-aided development”
- “CAD” has sometimes been erroneously equated with geometry

- These systems are built around the notion that the developer of a geometric model should construct the model to be consistent with the manufacturing process (**mCAD**)
- The analytical designer of a system wants to think about the function and performance of the device being generated, often leading to the generation of a separate **aCAD** model
- The modeling techniques supported by **aCAD** and **mCAD** are often so dissimilar that model transfer between them is done by limited translators or by “starting over”
- This one-way path from **aCAD** to **mCAD** leads to a “broken process”

- ESP is:
 - a geometry creation and manipulation system designed specifically to support the analysis and design of aerospace vehicles
 - can be run stand-alone for the development of models
 - can be embedded into other analysis and design systems to support their geometry needs
- ESP is not:
 - a full-featured computer-aided design (CAD) system designed specifically to support the mechanical design and manufacturing of any complex system
 - a system to be used for creating “drawings”





- Construction process guarantees that models are realizable solids
 - watertight representation needed for grid generators
 - sheets and wires are supported when needed
- Parametric models are defined in terms of:
 - Feature Tree
 - “recipe” for how to construct the configuration
 - Design Parameters
 - “values” that describe any particular instance of the configuration

- Configurations start with the generation of primitives
 - standard primitives: point, box, sphere, cone, cylinder, torus
 - grown primitives (from sketches): extrude, rule, blend, revolve, sweep, loft
 - user-defined primitives (UDPs)
- Bods can be modified
 - transformations: translate, rotate, scale, mirror
 - applications: fillet, chamfer, hollow
- Bods can be combined
 - Booleans: intersect, subtract, union
 - other: join, connect, extract, combine

```

# bolt example

# design parameters
1: DESPMTR  Thead  1.00  # thickness of head
2: DESPMTR  Whead  3.00  # width    of head
3: DESPMTR  Fhead  0.50  # fraction of head that is flat

4: DESPMTR  Dslot  0.75  # depth of slot
5: DESPMTR  Wslot  0.25  # width of slot

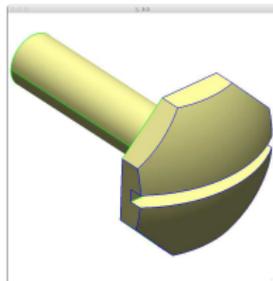
6: DESPMTR  Lshaft 4.00  # length  of shaft
7: DESPMTR  Dshaft 1.00  # diameter of shaft

8: DESPMTR  sfact  0.50  # overall scale factor

# head
9: BOX      0      -Whead/2 -Whead/2  Thead  Whead  Whead
10: ROTATEX 90 0 0
11: BOX      0      -Whead/2 -Whead/2  Thead  Whead  Whead
12: ROTATEX 45 0 0
13: INTERSECT

...

```



```

...
14: SET      Rhead  (Whead^2/4+(1-Fhead)^2*Thead^2)/(2*Thead*(1-Fhead))
15: SPHERE   0      0  0    Rhead
16: TRANSLATE Thead-Rhead  0  0
17: INTERSECT

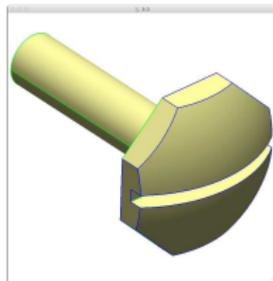
# slot
18: BOX      Thead-Dslot -Wslot/2  -Whead  2*Thead  Wslot  2*Whead
19: SUBTRACT

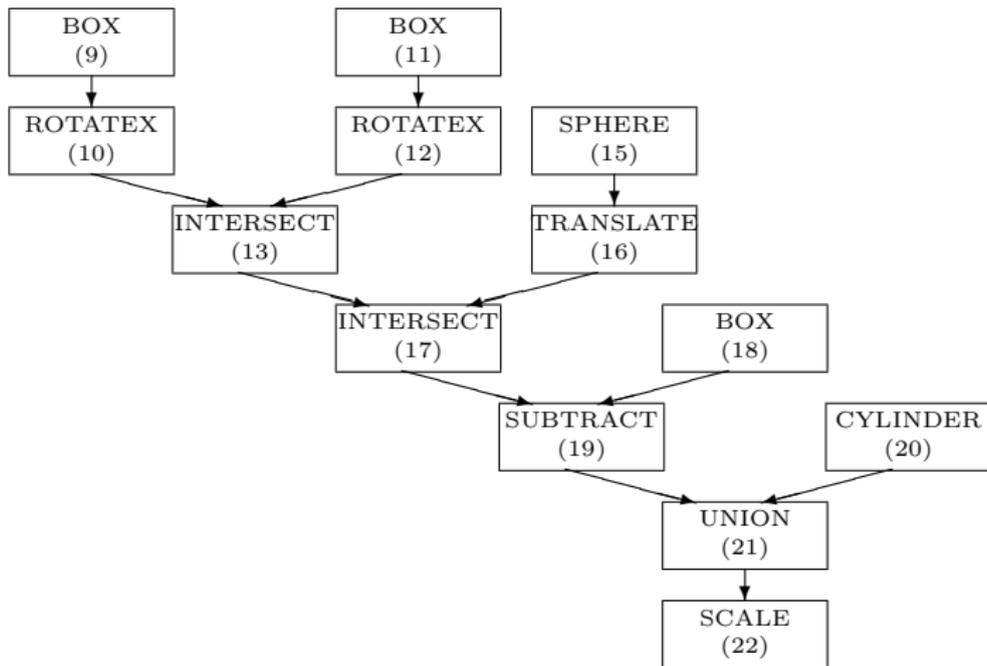
# shaft
20: CYLINDER -Lshaft  0  0  0  0  0  Dshaft/2
21: UNION

22: SCALE    sfact

23: END

```





- ESP models typically contain one or more Design Parameters
- Design Parameters can be single-valued, 1D vectors, or 2D arrays of numbers
- Each Design Parameter has a current value, upper- and lower-bounds, and a current “velocity” (which is used to define sensitivities)
- Design Parameters can be “set” and “get”
 - through ESP’s tree window
 - externally via calls to the Application Programming Interface (API)
- Arguments of all operations can be written as “expressions” that reference Design Parameters

- ESP maintains a set of global and local attributes on a configuration that are persistent through rebuilds
- Supports the generation of multi-fidelity models
 - attributes can be used to associate conceptually-similar parts in the various models
- Supports the generation of multi-disciplinary models
 - attributes can be used to associate surface groups which share common loads and displacements
- Supports the “marking” of Faces and Edges with attributes such as nominal grid spacings, material properties, ...

- ESP allows a user to compute the sensitivity of any part of a configuration with respect to any Design Parameter
- Many of OpenCSM's commands have been analytically “differentiated”
 - efficient, since there is no need to re-generate the configuration
 - accurate, since there is no truncation error associated with “differencing”
- Other commands (currently) require the use of finite-differenced sensitivities
 - robust, due to new mapping technique
 - less efficient, since it requires the generation of a “perturbed” configuration
 - less accurate, since one needs to carefully select a “perturbation step” that is a balance between truncation and round-off errors

- Users can add their own user-defined primitives (UDPs)
 - create a single primitive solid
 - are written in C, C++, or FORTRAN and are compiled
 - can be written either top-down or bottom-up
 - have access to the entire suite of methods provided by EGADS
 - are coupled into ESP dynamically at run time
- Users can add their own user-defined functions (UDFs)
 - consume one or more Bodys from stack
 - are otherwise similar to UDPs
- Users can add their own user-defined components (UDCs)
 - can be thought of as “macros”
 - create zero or more Bodys
 - are written as .csm-type scripts

- ESP's back-end (server) runs on a wide variety of modern compute platforms
 - LINUX
 - OSX
 - Windows
- ESP's user-interface (client) runs in most modern web browsers
 - FireFox
 - Google Chrome
 - Safari
 - Edge (chromium-based versions)
- ESP can be distributed anywhere in the computer environment
 - open-source project (using the LGPL 2.1 license) that is distributed as source

- Models are defined in `.csm` files
 - human readable ASCII
 - stack-like language that is consistent with Feature Tree traversal
 - contains looping via “patterns”
 - contains logical (if/then) constructs
 - contains error recovery via thrown/caught signals
- OpenCSM modeling system is defined by an Application Programming Interface (API) that allows it to be embedded into other applications
 - load a Master Model
 - interrogate and/or edit the Master Model
 - execute the Feature Tree and create BRep(s)
 - interrogate the BRep(s)
 - “set” and “get” sensitivities

- Double-clicking `runESP118` icon on desktop
 - Automatically starts server and brings up browser
 - User can select **File**→**Open** to use existing `.csm` file
 - Closing the browser automatically stops the server
 - No command-line options can be used
- Double-clicking on `ESP118` icon on desktop
 - Brings up a terminal window in which all the ESP environment variables are set
 - Allows user to launch `serveCSM` multiple times, with filenames and/or command-line options
 - Terminal window remains open until the user closes it

- If starting from terminal window:

- Technique 1: start browser automatically:

```
setenv ESP_START "open -a /Applications/Firefox.app ...  
... $ESP_ROOT/ESP/ESP.html"
```

OR

```
export ESP_START="open -a /Applications/Firefox.app ...  
... $ESP_ROOT/ESP/ESP.html"
```

OR

```
set ESP_START="open -a /Applications/Firefox.app ...  
... $ESP_ROOT/ESP/ESP.html"
```

and then

```
serveCSM $ESP_ROOT/data/tutorial1
```

- Technique 2: start browser separately:

```
serveCSM $ESP_ROOT/data/tutorial1
```

and then open a browser on ESP.html

- To start serveCSM

```
serveCSM [filename[.csm]] [options...]
```

where `filename` can be given in the following forms:

- (blank) starts without any input files (**File**→**Open** is then typically used within ESP)
- `name.csm` reads the given `.csm` file
- `name.cpc` reads the given `.cpc` file, which is a `.csm` file with all the UDCs inline
- `name.stp` or `name.step` or `name.STP` or `name.STEP` creates and reads `autoStep.csm` (which loads the given STEP file)
- `name.igs` or `name.iges` or `name.IGS` or `name.IGES` creates and reads `autoIges.csm` (which loads the given IGES file)
- `name.egads` or `name.EGADS` creates and reads `autoEgads.csm` (which loads the given EGADS file)
- otherwise a `.csm` extension is added and the file is read

- Frequently used [options...] include:
 - `-batch` runs the case but does not attach to a browser
 - `-help` or `-h` prints listing of acceptable options
 - `-jrn1 jrnlname` can be used to replay a previous session
 - current session is stored in file `portXXXX.jrn1`
 - file must be renamed to be used for next session
 - `-skipBuild` to skip initial build
 - `-skipTess` to skip tessellation at end (and automatically select `-batch`)
 - `--version` or `-version` or `-v` to return version information
 - ...

- Other [options...] include:
 - `-despmtrs despname` to update the Design Parameters from the `despname` file
 - `-dict dictname` loads Constant Parameters from the `dictname` file
 - `-dumpEgads` to dump EGADS file in form `Body_XXXXXX.egads` after each Body is built
 - `-loadEgads` to load `Body_XXXXXX.egads` file if it exists in current directory
 - `-onormal` to plot in (nearly) orthonormal (not perspective)
 - `-outLevel n` selects the output level (1 is the default)
 - `-port portnum` selects the port for communication with the browser (7681 is the default)
 - `-printStats` to print the contents of the stack after every command is executed (useful for debugging)

- Other [options...] include:
 - `-plot plotfile` to plot additional information or provide input for the `-histDist` or `-plugs` options
 - `-plotBDF filename` superimposes BDF information in Graphics Window
 - `-plotCP` to plot Bspline control points
 - `-sensTess` to produce configuration sensitivity (instead of tessellation sensitivity) output
 - `-histDist dist` to generate histograms of the points in the `plotfile` from the configuration. Points that are further than `dist` are added to a new `plotfile` called `bad.points`
 - `-plugs npass` to use the experimental Plugs program, in which the values of the Design Parameters are adjusted so as to minimize the distance of the points in `plotfile` from the configuration.

- Still other (less frequently used) [options...] include:
 - `-verify` to execute **ASSERT** statements that contain `verify=1`
 - `-addVerify` creates verification files (for automatic regression testing)
 - `-egg eggname` uses an external grid generator

- Other (for development) [`options...`] include:
 - `-checkMass` to compare the internally computed mass properties with those computed via the tessellation
 - `-checkPara` to check the parallelizability of the build
 - `-ptrb ptrbname` to generate information with which the sensitivities are debugged

- Faces
 - yellow — front of Face
 - grey — back of Face
 - black — grid
- Edges
 - green — manifold Edge that was first created as part of a primitive (such as the Edges in a BOX)
 - blue — manifold Edge that was first created as part of a Boolean or Applied Branch
 - brown — non-manifold Edge that supports only one Face
 - orange — non-manifold Edge that supports more than two Faces
 - black — grid
- Nodes
 - black

- Translation
 - press and drag any mouse button
- Rotation
 - hold down **Ctrl** and drag any mouse button
 - hold down **Alt** and drag any mouse button
- Zoom
 - hold down **Shift** and drag any mouse button
 - scrolling the middle mouse button also scrolls in/out
- Flying mode
 - press **!** in Graphics window to toggle mode
 - image continues moving image until mouse is released
- Note: the mouse mappings are defined in `ESP.js`

“flying-mode” is off by default

Key-press	“flying-mode” off	“flying-mode” on
←	rotate left 30°	translate left
→	rotate right 30°	translate right
↑	rotate up 30°	translate up
↓	rotate down 30°	translate down
+	zoom in	zoom in
-	zoom out	zoom out
PgUp	zoom in	zoom in
PgDn	zoom out	zoom out
Home	home view	home view

Note: holding **Shift** reduces the increment

Key-press	orientation	note
Ctrl-h	home view	y vs x
Ctrl-f	front view	y vs x
Ctrl-l	left side view	y vs z
Ctrl-r	right side view	y vs $-z$
Ctrl-b	bottom view	z vs x
Ctrl-t	top view	$-z$ vs x
Ctrl-i	zoom in	
Ctrl-o	zoom out	

Note: some of these signals may be intercepted by your browser

Button press	orientation	note
H	home view	y vs x
L	left side view	y vs z
R	right side view	y vs $-z$
B	bottom view	z vs x
T	top view	$-z$ vs x
+	zoom in	
-	zoom out	

Buttons are near top of Tree window

key press	action
>	save view (in memory)
<	restore view (from memory)
Ctrl->	save view (in a file)
.	save view (in a file)
Ctrl-<	restore view (from a file)
,	restore view (from a file)

- In the Tree window, **Display** contains an entry for each Body
- If the **Body** is expanded (the + on the left is pressed), then entries appear for **Faces**, **Edges**, **Nodes**, and **Csystems**
- If the **Faces**, **Edges**, **Nodes**, or **Csystems** are expanded, the names of all entities in the “group” are listed
- **Viz** toggles the visibility of the associated Body(s), Face(s), Edge(s), Node(s), or Csystem(s)
- **Grd** toggles the visibility of the grid of the associated Body(s), Face(s), or Edge(s)
- **Trn** toggles the pseudo-transparency of the associated Face(s)
- **Ori** toggles the orientation vectors of the associated Edge(s)
- Toggling at a “group” level effects the setting of its children
- Pressing **Display** gives the user the option of turning on/off the display of all Nodes, Edges, or Faces in all Bods

- Re-center the image at the current location and set a new “rotation center”
 - * or 8
- Find the location of the cursor (in 3D space) and report it in the Messages window
 - @ or 2
- Identify the object (Edge or Face) and list all its attributes in the Messages window
 - ^ or 6
- List the key-press options in the Messages window
 - ?
- Orientation of image in Graphics window
 - red axis in x -direction
 - green axis in y -direction
 - blue axis in z -direction

- Turn off the visibility of the Node, Edge, or Face at cursor
 - **v**
- Toggle the grid on the Edge or Face at cursor
 - **g**
- Toggle the transparency of the Face at cursor
 - **t**
- Toggle the orientation vectors of the Edge at cursor
 - **o**

- Show step-by-step build process
 - **StepThru** button (near top of Tree Window)
- Next step in build process
 - **NextStep** button (near top of Tree Window) or **n** key in Graphics Window
- Previous step in build process
 - **p** key in Graphics Window
- First step in build process
 - **f** key in Graphics Window
- Last step in build process
 - **l** key (letter “l”) in Graphics Window
- Exit StepThru mode
 - **CancelStepThru** at bottom of Display listing in Tree Window

- Method:
 - start ESP: `serveCSM`
 - add Design Parameter by pressing **DesignParameters**
 - add Branch by pressing **Branch**
- Advantages:
 - most similar to other CAD packages
 - can use interactive sketcher
- Disadvantages:
 - generally slow
 - cannot add comments, indentation, etc.
 - harder to debug

- Method:
 - use any text editor to create `myFile.csm`
 - run ESP: `serveCSM -loadEgads -dumpEgads myFile`
- Advantages;
 - can use any editor with which you are familiar
 - easy to add comments, spacing, indentation, ...
- Disadvantages:
 - do not get help in writing `.csm` file
 - cannot use interactive sketcher
 - requires many ESP restarts

- Method:
 - start ESP: `serveCSM`
 - **File**→**Edit** and then **Save**
- Advantages:
 - context-sensitive editor with hints
 - easy to add comments, spacing, indentation, ...
- Disadvantages:
 - slightly different key mappings
 - cannot use interactive sketcher



Using the jrnl (1)

- Every time that you execute **ESP**, a new `.jrnl` file is generated (which overwrites any existing file)
 - default name if `port7681.jrnl` (unless you used the `-port` command line option)
- The `.jrnl` file remembers all the interactions that you had with the ESP interface (example on next page)
- Each user action is a separate line in the `.jrnl` file



Using the jrnl (2)

Example port7681.jrnl

```
setPmtr|H|1|1|3|  
build|0|  
clrVels|  
setVel|D|1|1|1|  
build|0|
```

- To use a `.jrnl` file, follow these steps:
 - when ESP completes, rename the `.jrnl` file, with a command such as

```
mv port7681.jrnl my.jrnl
```

or

```
ren port7681.jrnl my.jrnl
```

(this is needed so that the `.jrnl` is not overwritten below)

- edit the `.jrnl` file to remove the offending command (which is usually the last line)
- restart ESP with the command

```
serveCSM -jrnl my.jrnl my.csm
```

(assuming that the name of your `.csm` file is `my.csm`)

- ESP has two ways of saving your work:
 - **File**→**Edit**→**Save**
 - Save an exact copy of information in the code editor
 - Remembers comments, indentation, line-splitting, spacing, etc.
 - Is preferred method of saving your work, unless you make changes in the ESP Tree Window (for example, add/edit/remove a Branch or change a Design Parameter)
 - **File**→**Export FeatureTree**
 - Makes an output file by reading the current feature tree
 - Forgets comments, indentation, line-splitting, spacing, etc.
 - Is only useful if you have made edits via the Tree Window



Saving vs. Exporting (2)

Original .csm file

```
# example program
# written by John Dannenhoffer

# define parameters for the box
DESPMTR  L   3.0  # length (ft)
DESPMTR  H   2.0  # height (ft)
DESPMTR  D   1.0  # depth  (ft)

# create the box (centered at the origin)
BOX      -L/2  -H/2  -D/2 \
         L     H     D

# put _name attributes on the Faces
PATBEG   iface  6
        SELECT  FACE  iface
        ATTRIBUTE _name $face_+iface
PATEND

END
```



Saving vs. Exporting (3)

.csm file generated by Export FeatureTree

```
# example_out.csm written by ocsmsave (v1.18)

# Constant, Design, and Output Parameters:
despmtr  L      3.00000
despmtr  H      2.00000
despmtr  D      1.00000

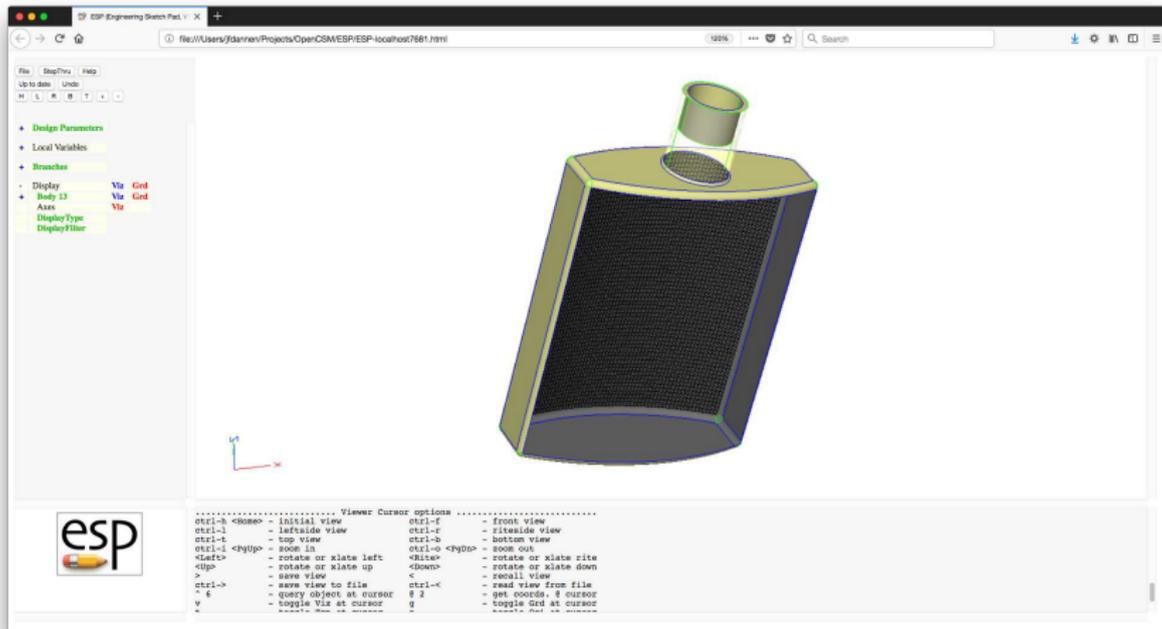
# Global Attributes:

# Branches:
box      -L/2  -H/2  -D/2  L  H  D
patbeg   iface  6
        select  FACE  iface
attribute _name  $face_+iface
patend

end
```


- If the Message Window turns yellow
 - `OpenCSM` has detected an error
 - Double-clicking in the Message Window will automatically open the code editor to the appropriate line
- If the Message Window turns pink
 - `ESP` has lost its connection to `serveCSM` and the session must be restarted
 - Consider using the `-jrn1` option to get you (almost) back to the situation that caused the connection to be lost

- 1 Start `serveCSM` using the file
`$ESP_ROOT/training/ESP/data/session01/bottle2.csm`
or
`../training/ESP/data/session01/bottle2.csm`
 - Note that on Windows, you will need to use backslash (`\`) instead of the forward slash (`/`)
- 2 Explore the various image manipulation tools
- 3 See if you can get the image on the next page
- 4 Use `StepThru` to see how the bottle was created



- Opportunity to provide immediate “feedback”
- Any questions about presentation material, critique of sample problems, ...
- Mail questions to jfdannen@syr.edu
- Questions will be answered at next session