

Engineering Sketch Pad (ESP)



Training Session 6 UDPs, UDFs, and UDCs

John F. Dannenhoffer, III

jfdannen@syr.edu
Syracuse University

Bob Haimes

haimes@mit.edu
Massachusetts Institute of Technology
updated for v1.22

- User-defined Primitives (UDPs) and Functions (UDFs)
 - Difference Between UDPs and UDFs
 - Using UDPARG and UDPRIM Statements
- Creating Simple Cross-sections
- Creating a simple NodeBody, WireBody, SheetBody, and SolidBody
- User-defined Components (UDCs)
 - Include-style
 - Function-style
- Homework Exercise

- Users can add their own user-defined primitives (UDPs)
 - creates a single Body
 - do not consume any Bodys from the Stack
 - are written in C, C++, or FORTRAN and are compiled
 - can be written either top-down or bottom-up or both
 - 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)
 - are the same as UDPs, except they consume one or more Bodys from the Stack

- UDPs are called with a UDPRIM statement

```
UDPRIM      $primetype $argName1 argValue1 \  
             $argName2 argValue2 \  
             $argName3 argValue3 \  
             $argName4 argValue4
```

- \$primetype must start with a letter
- At most 4 name-value pairs can be specified on the UDPRIM statement
- More name-value pairs can be specified in any number of UDPARG statements that precede the UDPRIM statement

```
UDPARG      $primetype $argName1 argValue1 \  
             $argName2 argValue2 \  
             $argName3 argValue3 \  
             $argName4 argValue4
```

- name-value pairs are processed in order (with possible over-writing)

- For UDPs that read an external file, one can use << to tell ESP to create a file from the following lines, up to a line that starts with >>
- For example:

```
UDPRIM      editAttr  filename <<  verbose 1
            NODE ADJ2FACE tagType=spar tagIndex=1
            AND   ADJ2FACE tagType=lower
            AND   ADJ2EDGE tagType=root
            SET                      capsConstraint=pointConstraint1
>>
SET          A    10
```

has two Branches (UDPRIM and SET)

- The following generate identical Boxes

```
UDPRIM box dx 1 dy 2 dz 3
```

- and

```
UDPARG box dx 1
```

```
UDPRIM box dy 2 dz 3
```

- and

```
UDPARG box dx 11 dy 22 dz 33
```

```
UDPRIM box dx 1 dy 2 dz 3
```

- and

```
UDPARG box dx 1
```

```
UDPARG box dy 2
```

```
UDPARG box dz 3
```

```
UDPRIM box
```

- Some UDPs return values to the calling script
- The returned values have names that are prepended by two at-signs (for example: `volume` in the UDP is available as `@@volume` after the UDPRIM executes)
- These values stay in effect until overwritten by another UDP (or a UDF or a UDC)

- `bezier $filename debug=0 @@imax @@jmax cp[]`
 - generate a Bezier WireBody, SheetBody, or SolidBody from an input file
- `biconvex thick=0 camber=0`
 - generate a biconvex airfoil SheetBody
- `box dx=0 dy=0 dz=0 rad=0 @@area @@volume`
 - generate a (rectangular) WireBody, SheetBody, or SolidBody centered at the origin (with possibly-rounded corners)
- `csm $filename $pmtrname pmtrvalue=0 @@volume`
 - call `OpenCSM` recursively to read a `.csm` file and create a Body
- `ellipse rx=0 ry=0 rz=0 nedge=2 thbeg=0`
 - generate an ellipse SheetBody centered at the origin (try to use the `supell` UDP instead)
- `fitcurve $filename ncp ordered periodic xform[]`
`@@npnt @@rms`
 - fit a Bspline curve WireBody to a set of points

- `freeform $filename imax=1 jmax=1 kmax=1 xyz[]`
 - generate a freeform WireBody, SheetBody, or SolidBody from an input file
- `hex corners[] uknots[] vknots[] wknots[] @@area @@volume`
 - create a general hexahedron SolidBody from its corners segments
- `import $filename bodynumber=1 @@numbodies`
 - read a Body (or Bodys) out of a .step file
- `kulfan class[] ztail[] aupper[] alower[]`
 - generate a Kulfan SheetBody airfoil
- `naca series=0012 thickness=0 camber=0 maxloc=0.4 offset=0 sharpte=0`
 - generate a NACA 4-series SheetBody airfoil or WireBody camberline

- `naca456 thkcode toc xmaxt leindex camcode cmax xmaxc cl a`
 - generate a NACA 4-, 5-, or 6-series SheetBody airfoil
- `nurbbody $filename`
 - generate a Body from a series of NURBS
- `parsec yte poly[] param[] meanline ztail[]`
 - generate a Parsec SheetBody airfoil by either specifying Sobieski's parameters or spline parameters
- `pod length=0 fineness=0 @@volume`
 - generates a VSP-like SolidBody pod
- `poly points[]`
 - generate a general SolidBody polyhedron, SheetBody polygon, WireBody line, or NodeBody point

- `prop nblade cpower lambda reyr rtip rhub clift
cdrag alfa shdiam=0 shxmin shxmax spdiam=0 spxmin
@@cthrust @@eff`
 - generates a propeller and optional shaft and spinner
- `radwaf ysize=0 zsize=0 nspoke=0 xframe[]`
 - generate a radial SheetBody waffle, which is useful for creating fuselage structures
- `sample dx dy dz center[] @@area @@volume`
 - used as an example for users who want to create their own UDP
- `sew $filename toler=0 bodynum=1`
 - sew Faces in a `step` file into a SolidBody

- `stag rad1=0.1 beta1=30 gama1=10 rad2=0.05 beta2=-40
gama2=5 alfa=-30 xfrnt=0.333 xrear=0.667`
 - simple turbomachinery airfoil generator to generate a SheetBody
- `supell rx rx_w rx_e ry ry_s ry_n n n_w n_e n_s n_n
n_sw n_se n_nw n_ne offset nquad`
 - generate a 4-quadrant SheetBody super-ellipse
- `waffle depth=1 segments[] $filename progress=0`
 - generate a SheetBody waffle by extruding a 2D group of segments

```
# naca
```

```
UDPRIM naca thickness 0.00 camber 0.04  
TRANSLATE -2 0 0
```

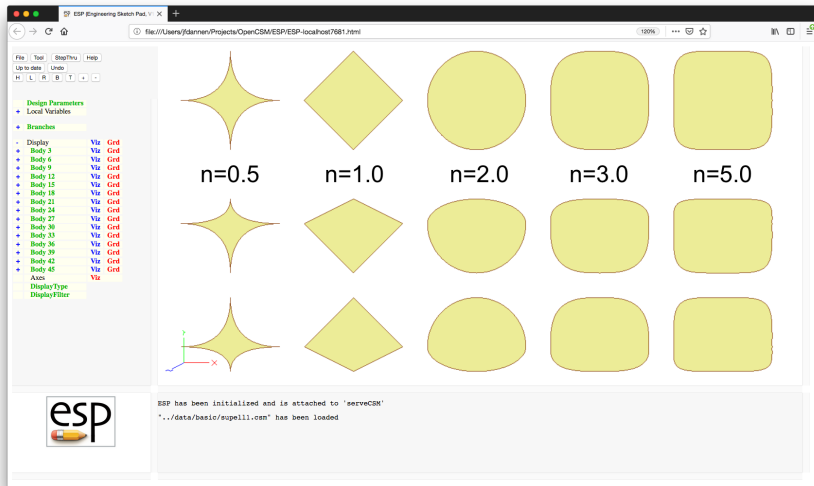
```
UDPRIM naca thickness 0.12 camber 0.00
```

```
UDPRIM naca thickness 0.12 camber 0.04  
TRANSLATE +2 0 0
```

```
END
```



Generated with `$ESP_ROOT/data/basic/supell1.csm`



```
# simple
```

```
POINT      -3 0 0
```

```
UDPRIM box  dy 1.0
```

```
TRANSLATE -1 0 0
```

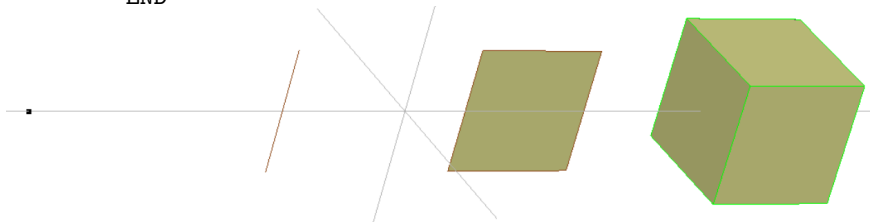
```
UDPRIM box  dx 1.0  dy 1.0
```

```
TRANSLATE +1 0 0
```

```
UDPRIM box  dx 1.0  dy 1.0  dz 1.0
```

```
TRANSLATE +3 0 0
```

```
END
```



- `createBEM $filename space=0 imin=3 imax=5 nocrod=0`
 - create a NASTRAN-type built-up-element (BEM) file from Body on Stack
- `createPoly $filename hole[]`
 - create a TETGEN .poly file between the two BODYS on the top of the Stack
- `droop xle=-100 thetale=0 xte=100 thetate=0`
 - applies leading- or trailing-edge droop to the Body on the top of the Stack
- `editAttr $attrname $input $output overwrite=0 $filename verbose=0 @@nchange`
 - edit the Attributes for the Body on the top of the Stack
- `flend slopea=1 slopeb=1 toler=1e-6 equis=0 npnt=33 plot=0`
 - create a flend (similar to fillet) that connects the one or two BODYS on the top of the Stack

- `ganged $op toler=0`
 - perform ganged SUBTRACTs or UNIONs to Bodys on the Stack back to the Mark
- `guide nxsect=5 origin=0 axis=0`
 - sweep a SheetBody or WireBody along a WireBody guide curve
- `matchBodys toler @@nnodes @@nedges @@nfaces`
- `nuscale xscale=1 yscale=1 zscale=1 xcent=0 ycent=0 zcent=0`
 - converts Body on top of stack to BSplines and applies separate scaling in each coordinate direction
- `printBbox`
 - print the bounding boxes associated with the Bodys on the Stack

- `printBrep`
 - print Brep information associated with the Bodys on the Stack
- `printEgo`
 - print EGO information associated with the Bodys on the Stack
- `slices nslice dirn=0`
 - creates uniform slices of Body on top of stack
- `stiffener beg[] end[] depth=0 angle=0`
 - create a stiffener that is orthogonal to the SheetBody on the top of the Stack
- UDFs are called in exactly same way as UDPs are called

- see `EngSketchPad/doc/UDP_UDF/udp_udf.pdf`

- A UDC is a series of statements that are contained in a `.udc` file
- The statements in the UDC can be treated in two ways:
 - Include-style
 - statements within the UDC are simply processed as if they were included in the enclosing `.csm` or `.udc` file
 - the `.udc` file must start with an `INTERFACE . ALL` statement
 - Variables and Parameters in the `.udc` file have the same scope as its caller (that is, the UDC shares variables with its caller)
 - Function-style
 - Variables and Parameters in the `.udc` file have local scope (that is, the UDC's variable are private)
 - Variables in the UDC get values via `INTERFACE . IN` statements
 - The UDC can output some of its variables via `INTERFACE . OUT` statements

- In test1.csm

```
SET      A      1
SET      B      10
SET      C      0
UDPRIM   $/test2
SET      D      C^2
```

- In test2.udc

```
INTERFACE .    ALL
SET              C  A+B
```

- After running, C=11 and D=121

- In test3.csm

```
SET      A      1
SET      B      10
SET      C      0
UDPRIM   $/test4  first A   second B
SET      D      C^2
```

- In test4.udc

```
INTERFACE first IN 0
INTERFACE second IN 0
INTERFACE sum OUT 0
SET      C      999
SET      sum     first+second
```

- After running, C=0, D=0, and @@sum=11

- `applyTparams factor=1`
 - apply `.tParams` to the Edges and Faces of the Body on the top of the Stack
- `biconvex thick=0`
 - generate a biconvex airfoil
- `boxudc dx=0 dy=0 dz=0 @@vol`
 - similar to the box UDP
- `contains @@contains`
 - determine if either of the two Bodys on the top of the Stack contains the other
- `diamond thick=0`
 - generate a double-diamond airfoil
- `duct diameter=1 length=2 thickness=0.10 camber=0.04`
 - generate a duct

- expressions `xx yy zz @@aa @bb`
 - a test UDC that has no other practical use
- `flapz xflap yflap theta=15 gap=0.01 openEnd=0`
 - cut a (deflected) flap in a Body
- `fuselage xloc zloc width height noselist taillist`
 - generate a fuselage
- `gen_rot xbeg=0 ybeg=0 zbeg=0 xend=1 yend=1 zend=1
rotang=0 @@azimuth @@elevation`
 - general rotation with two fixed points
- `overlaps @@overlaps`
 - determine if the two Bodys on the top of the Stack overlap each other
- `popupz xbx ybax height=1`
 - pop up a part of the configuration

- spoilerz xbox ybox depth=1 thick=0.1 theta=30 overlap=0.002 extend=0.20
 - pop up a spoiler
- strut length=2.0 thickness=0.2 height=1.0 sweep=0
 - generate a strut (between a duct and wing)
- swap
 - swaps the two Bodys or Marks on the top of the stack
- wake mirror=0 area=100 aspect=8 taper=0.8 twist=-5 sweep=0 dihedral=0 camber=0.04 wakeLen=3.0 wakeAng=0
 - generate a wake
- wing mirror=0 area=100 aspect=8 taper=0.8 twist=-5 sweep=0 dihedral=0 thickness=0.12 sharpte=0 camber=0.04 inboard=0 outboard=1 pctchord=0 angleleft=0 angrite=0 spar1=0 spar2=0 nrrib=0 @@span
 - generate a wing

- UDCs are called with a UDPRIM statement
- \$primetype must start with a slash (/), dollar-slash (\$/), or dollar-dollar-slash (\$\$/)
 - if /, then the UDC file is in the current working directory
 - if \$/, then the UDC file is in the same directory as the .csm file
 - if \$\$\$/, then the UDC file is in ESP_ROOT/udc
- The UDPRIM statement can be preceded by one or more UDPARG statements
- name-value pairs are processed in order (with possible over-writing)

- Define the interface
 - input variables (with default values)
 - output variables (with default values)
 - dimensioned variables (which all default to 0)
- Add assertions to ensure valid inputs
- Make sure all “output” variables are assigned values

```
# make sure that there are at least entities on the Stack
IFTHEN @stack.size LT 2
    THROW 999 # not enough entries on Stack
# if Mark,Mark on top of Stack
ELSEIF @stack[@stack.size-1] EQ 0 AND @stack[@stack.size] EQ 0
# if Body,Mark on top of Stack
ELSEIF @stack[@stack.size] EQ 0
    STORE .
    STORE tempSwap 99
    MARK
    RESTORE tempSwap 99
# if Mark,Body on top of Stack
ELSEIF @stack[@stack.size-1] EQ 0
    STORE tempSwap 99
    STORE .
    RESTORE tempSwap 99
    MARK
# if Body,Body on top of Stack
ELSE
    STORE tempSwap 98
    STORE tempSwap 99
    RESTORE tempSwap 98
    RESTORE tempSwap 99
ENDIF
```

```
# dumbbell

INTERFACE Lbar      in  0      # length of bar
INTERFACE Dbar      in  0      # diameter of bar
INTERFACE Dball     in  0      # diameter of balls
INTERFACE vol       out 0      # volume

ASSERT      ifpos(Lbar,1,0)    1
ASSERT      ifpos(Dbar,1,0)    1
ASSERT      ifpos(Dball,1,0)   1
SET         Lhalf      "Lbar / 2"

CYLINDER    -Lhalf  0  0  +Lhalf  0  0  Dbar
SPHERE      -Lhalf  0  0    Dball
UNION
SPHERE      +Lhalf  0  0    Dball
UNION

SET         vol        @volume

END
```

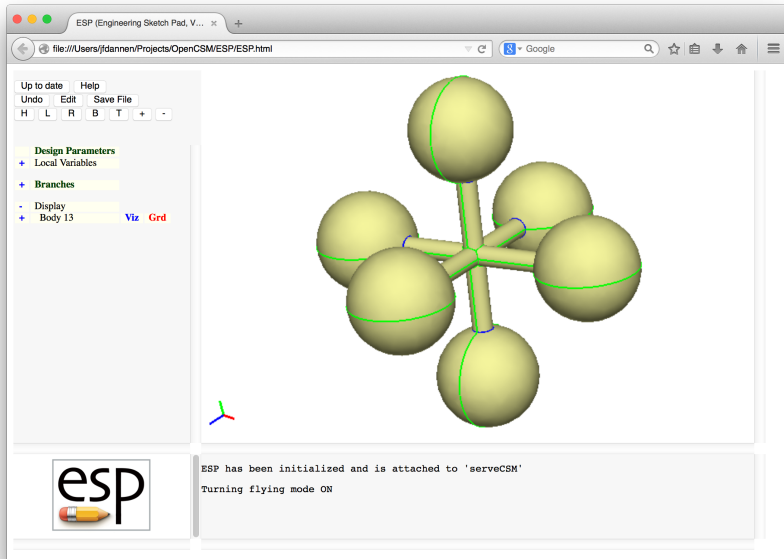
```
# jack

UDPARG $/dumbbell Lbar 5.0
UDPARG $/dumbbell Dball 1.0
UDPRIM $/dumbbell Dbar 0.2
SET     foo @@vol
STORE   dumbbell 0 1

RESTORE dumbbell
ROTATEY 90 0 0
UNION

RESTORE dumbbell
ROTATEZ 90 0 0
UNION

# show that vol was a local variable in .udc
ASSERT ifnan(vol,1,0) 1
END
```



```
# cutter
```

```
INTERFACE xx      in  0
INTERFACE yy      in  0
INTERFACE zbeg    in  0
INTERFACE zend    in  0
```

```
ASSERT    ifpos(xx.size-2,1,0)  1
ASSERT    ifzero(xx.size-yy.size,1,0)  1
```

```
SKBEG      xx[1]      yy[1]      zbeg
  PATBEG i xx.size-1
    LINSEG  xx[i+1]    yy[i+1]    zbeg
  PATEND
  LINSEG    xx[1]      yy[1]      zbeg
SKEND  1
```

```
EXTRUDE    0  0  zend-zbeg
```

```
END
```

```
# scribeCyl

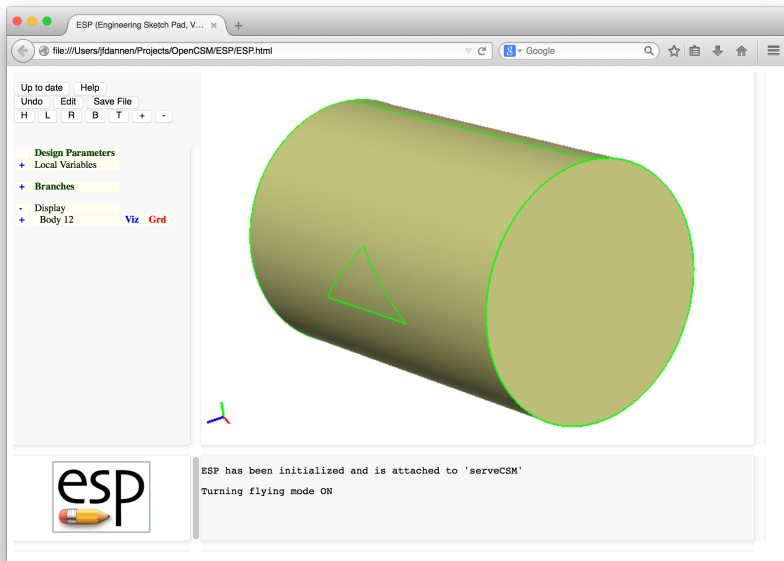
DIMENSION xpoints 1 3
DIMENSION ypoints 1 3

SET      xpoints "-1.; 1.; .0;"
SET      ypoints "-.5; -.5; +.5;"

CYLINDER -3 0 0 +3 0 0 2
ROTATEX  90 0 0

UDPARC  $/cutter xx    xpoints
UDPARC  $/cutter yy    ypoints
UDPARC  $/cutter zbeg  0
UDPRIM  $/cutter zend  3
SUBTRACT

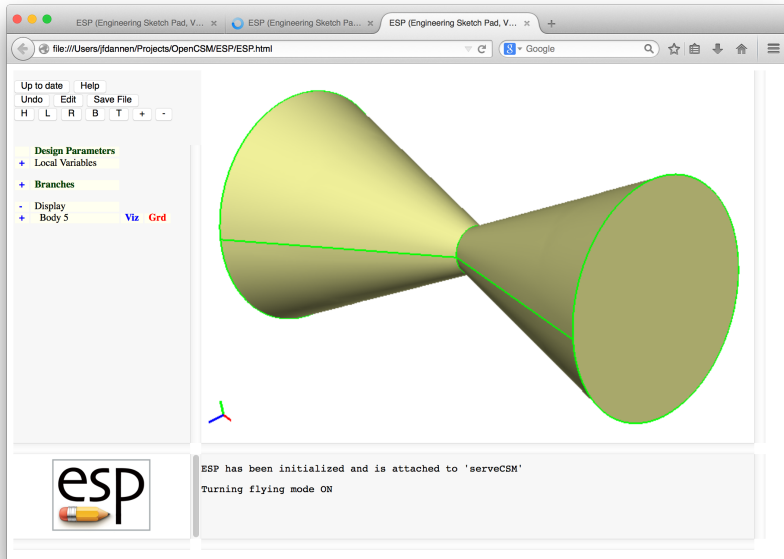
END
```





Homework Exercises

- Reflected cone
- Files in `$ESP_ROOT/training/ESP/data/session06` will get you started



- Write `mirrorDup.udc` to
 - store a copy of the Body on the top of the Stack
 - mirror the Body across a plane whose normal vector and distance from the origin are given
 - union the original and mirrored Bodys
- Apply `mirrorDup.udc` to a cone
 - cone base at $(5, 0, 0)$
 - cone vertex at $(0, 0, 0)$
 - cone diameter is 4
 - reflection across a plane at $x = 1$