

Engineering Sketch Pad (ESP) Training

Session 5: Using UDPs and UDCs

John F. Dannenhoffer, III

Syracuse University

Bob Haimes

Massachusetts Institute of Technology

Revised for v1.08





Overview

- Difference between UDPs and UDCs
- Using UDPs
 - list of UDPs shipped with ESP
 - calling a UDP
- Using UDCs
 - list of UDCs shipped with ESP
 - calling a UDC
- Writing a UDC
 - creating the interface
 - example UDC
- Hands-on exercises:
 - reflected cone
 - fuselage



Differences Between UDPs and UDCs

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



UDPs Shipped with ESP (1)

- `bezier` — generate a Bezier Wire, Sheet, or Solid Body from a input file
- `biconvex` — generate a biconvex airfoil
- `box` — generate a (rectangular) Wire, Sheet, or Solid Body centered at the origin (with possibly-rounded corners)
- `ellipse` — generate an ellipse centered at the origin
- `freeform` — generate a freeform Wire, Sheet, or Solid Body from an input file
- `import` — read a Body out of a `.step` file
- `kulfan` — generate a Kulfan airfoil segments



UDPs Shipped with ESP (2)

- `naca` — generate a NACA 4-series airfoil or camberline
- `naca456` — generate a NACA 4-, 5-, or 6-series airfoil
- `parsec` — generate a Parsec airfoil by either specifying Sobieski's parameters or spline parameters
- `sew` — sew Faces in a step file into a Solid Body
- `supell` — generate a 4-quadrant super-ellipse
- `waffle` — generate a waffle by extruding a 2D group of segments



Calling a UDP

- 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 pair can be specified on the udprim statement
- More name-value pairs can be specified in any number of updarg statements that precede the udprim statement

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

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



UDCs Shipped with ESP

- `biconvex` — generate a biconvex airfoil
- `boxudc` — similar to the box UDP
- `diamond` — generate a double-diamond airfoil
- `flapz` — cut a (deflected) flap in a Body
- `gen_rot` — general rotation with two fixed points
- `popupz` — pop up a part of the configuration
- `spoilerz` — pop up a spoiler
- `duct` — generate a duct
- `fuselage` — generate a fuselage
- `strut` — generate a strut (between a duct and wing)
- `wing` — generate a wing



Calling a UDC

- UDCs are called with a `udprim` statement
- `$primetype` must start with a slash (/) or 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
- The `udprim` statement can be preceded by one or more `udparg` statements
- name-value pairs are processed in order (with possible over-writing)



Writing a UDC

- 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



Example UDC — dumbbell.udc

```
# 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
```



Example UDC — jack.csm

```
# jack

udparg $/dumbbell Lbar 5.0
udparg $/dumbbell Dball 1.0
udprim $/dumbbell Dbar 0.2
set     foo @@vol
store   dumbbell 0 1

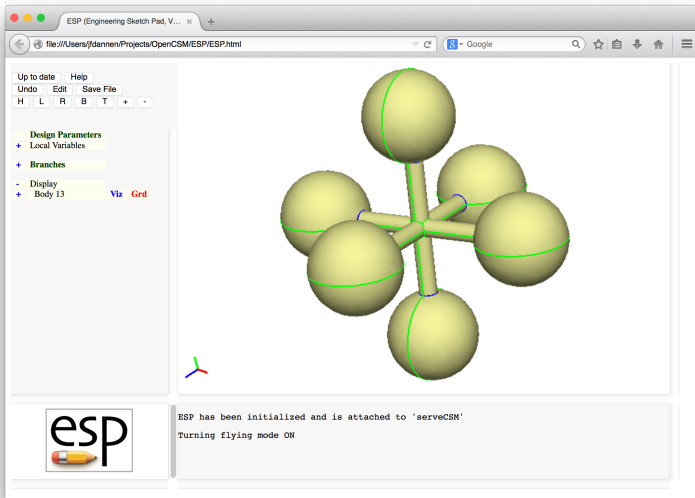
restore dumbbell
rotatey 90 0 0
union

restore dumbbell
rotatez 90 0 0
union

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



Example UDC — Jack





Example UDC — cutter.udc

```
# 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
```



Example UDC — scribeCyl.csm

```
# scribeCyl

dimension xpoints 1 3
dimension ypoints 1 3

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

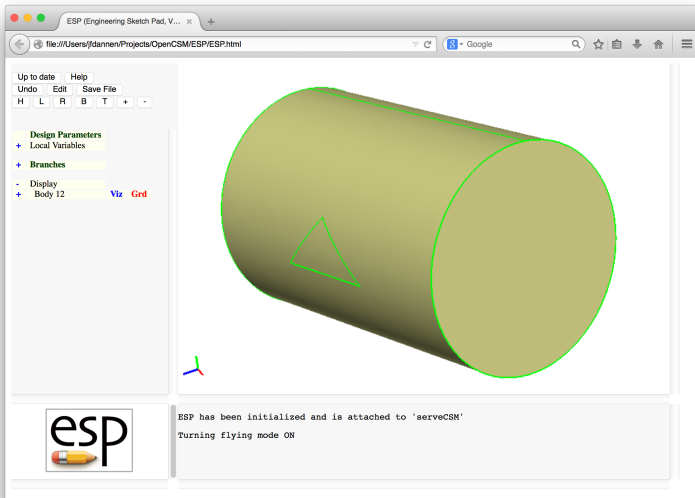
cylinder -3 0 0 +3 0 0 2
rotatex  90 0 0

udparg   $/cutter xx    xpoints
udparg   $/cutter yy    ypoints
udparg   $/cutter zbeg  0
udprim   $/cutter zend  3
subtract

end
```



Example UDC — Scribed Cylinder



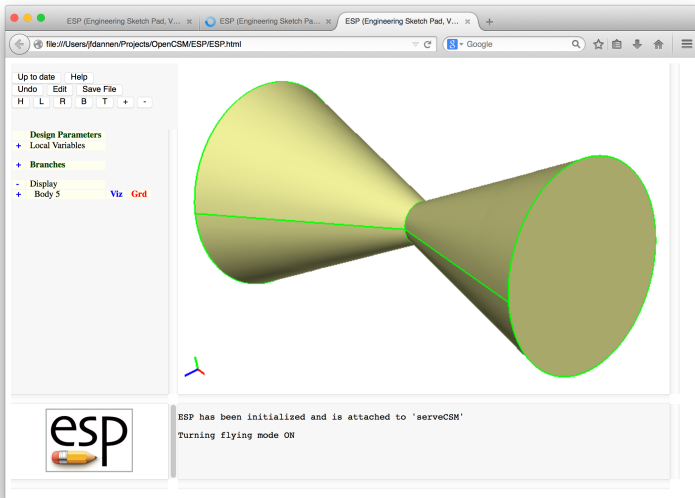


Hands-on Exercises

- 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 Bods
- Apply `mirrorDup.udc` to a cone whose vertex is at the origin
- Write `fuselage.udc` to create a fuselage by blending a series of super-ellipses, where the dimensions of the cross-sections are provided in arrays



Reflected Cone





Muddy Cards

- Any questions?
- Any suggestions?
- Were the examples useful?