Engineering Sketch Pad (ESP)

Training Session 10
Putting It All Together

John F. Dannenhoffer, III
jfdannen@syr.edu
Syracuse University

Bob Haimes
haimes@mit.edu
Massachusetts Institute of Technology

updated for v1.22

Multi-X Models

- During the design of an aircraft, various coupled models are needed
  - different disciplines
    - structures
    - controls
    - aerodynamics
    - ...
  - different fidelities
    - conceptual design
    - preliminary design
    - detailed design
- There needs to be communication between these models

Computational Aircraft Prototype Syntheses (CAPS)

- In order to support multi-fidelity and multi-disciplinary analyses, the CAPS program has been developed
  - funded by the AFRL
- CAPS uses geometries (and sensitivities) generated by ESP
- CAPS provides interfaces to many analysis programs, including:
  - aerodynamics (at various fidelities)
  - structures (at various fidelities)
  - ...
- There is a companion training course for CAPS that can be offered if there is sufficient interest

Multiple Views

- One of the strengths of ESP is to be able to have multiple "views" of a single configuration
  - tailored to a specific analysis method
  - driven by a single set of Design Parameters
  - attributed so that "common" features could be linked together
- Biggest problem is that such models can get very large
  - break up into nested user-defined compnents (UDCs)

High level UDCs

- Control — top level
  - `transport.csm` — definition of various "views"
  - `transport_init` — definition of various components
- Views — specific geometric model for a specific AIM, including necessary `CAPS` Attributes
  - `viewConcept` — conceptual view, useful to understand interactions
  - `viewVlm` — vortex lattic method (such as `AVL`)
  - `viewCfdInviscid` — outer-model line, including deflected controls, for CFD analyses (such as `SU2` or `fun3d`)
  - `viewCfdViscous` — outer-model line, including free-flying controls, for CFD analyses (such as `SU2` or `fun3d`)
  - `viewBem` — built-up element model, for use by structural solvers (such as `ASTROS` or `NASTRAN`)

UDCs for each Component — 1

Does not include any reference to `CAPS`

- Models — geometric models, created by various combinations of the primitives
  - `wingVlm` — cross-sectional cuts
  - `wingCfdInviscis` — outer mold line, including deflected control surfaces
  - `wingCfdViscous` — outer mold line, including free-flying control surfaces
  - `wingBem` — built-up element model, built by intersecting a waffle with the wing shape
- Primitives — lowest-level geometries
  - `wingOml` — outer mold line
  - `wingWaffle` — arrangement of spars and ribs
  - `wingHinges` — location of hinge lines for control surfaces

UDCs for each Component — 2

Does not include any reference to `CAPS`

- Initialization
  - `wingPmtrs` — definition of CFGPMTRs and DESPMTRs
  - `wingCalc` — high-level values computed from the CFGPMTRs and DESPMTRs

Basic Rules for Initializations

- `wingPmtrs`
  - Written as include-type UDC (`INTERFACE . ALL`)
  - Contains only `DIMENSION`, `CFGPMTR`, and `DESPMTR` statements
- `wingCalcs`
  - Written as include-type UDC (`INTERFACE . ALL`)
  - Contains only `OUTPMTR` and `SET` statements (for values that are useful by other components)

Basic Rules for Model and Primitives

- Knows nothing about `CAPS`
- Written as an include-type UDC (`INTERFACE . ALL`)
- Returns immediately if Body already exists
- Builds all subordinate models and primitives
- Puts all Bodys into one STOREd Group (with the name matching the model or primitive name)
- Leaves stack the same as it was upon entry

Basic Rules for Views

- Written as an include-type UDC (`INTERFACE , ALL`)
- Builds necessary Groups
- RESTOREs each Group
  - add view-specific (`CAPS`) Attributes
  - provide a `_name`
- Leaves all RESTOREd Bodys on the stack

Use of Unit Tests

- As each new `.udc` file is written, a unit test should be created

```
# wingVlm (test driver)
# written by John Dannenhoffer

UDPRIM    $/../transport_init

# make and show the wingVlm
UDPRIM    $/../wingVlm
RESTORE         wingVlm

END
```

Adding a New "view"

New view will be called vvv

- Create `viewVvv.udc` and `unittest/viewVvv.csm`
- Edit `transport.csm`
  - add

    ```
    CFGPMTR    VIEW:Vvv 1
    ```

  - add

    ```
    IFTHEN     VIEW:Vvv NE 0
       UDPRIM     $/viewVvv
    ENDIF
    ```

Adding a new component — 1

New component will be called `ccc`

- Create `cccPmtrs.udc` that contains all CFGPMTRs and DESPMTRs
- Create `cccCalc.udc` that contains all top-level values
- Edit `transport_init.udc`
  - add

    ```
    CFGPMTR    COMP:ccc 1
    ```
  - add

    ```
    UDPRIM     $/cccPmtrs
    ```
  - add

    ```
    UDPRIM     $/cccCalc
    ```

Adding a new component — 2

New component will be called `ccc`

- For each primitive (called `ppp`)
  - create `cccPpp.udc` and `unittest/cccPpp.csm`
- For each model (called `mmm`)
  - create `cccMmm.udc` and `unittest/cccMmm.csm`
- For each view file
  - add

```
IFTHEN    COMP:ccc NE 0
   UDPRIM    $/cccMmm
ENDIF
```

  - add

```
IFTHEN    COMP:ccc NE 0
  << build necessary view >>
ENDIF
```

Build Up Config. in Multiple Versions — 1

Note: a `unittest` file should be created for every starred `.udc` file

1. add `transport.csm`, `transport_init.udc`, `wingPmtrs.udc`, `wingCalc.udc`, `wingOml.udc*`, and `viewConcept.udc*`
2. add `viewVlm.udc*`
3. add `wingHinges.udc*`
4. add `viewCfdInviscid.udc*`
5. add `wingWaffle.udc*` and `wingBem.udc*`, `viewBem.udc*`
6. add `htailPmtrs.udc`, `htailCalc.udc`, `htailOml.udc*`, `htailHinges.udc*`, and tt htailVlm.udc*
7. add `htailWaffle.udc*`, `htailBem.udc*`
8. add `vtailPmtrs.udc`, `vtailCalc.udc`, `vtailOml.udc*`, `vtailHinges.udc*`, and tt vtailVlm.udc*
9. add `vtailWaffle.udc*`, `vtailBem.udc*`

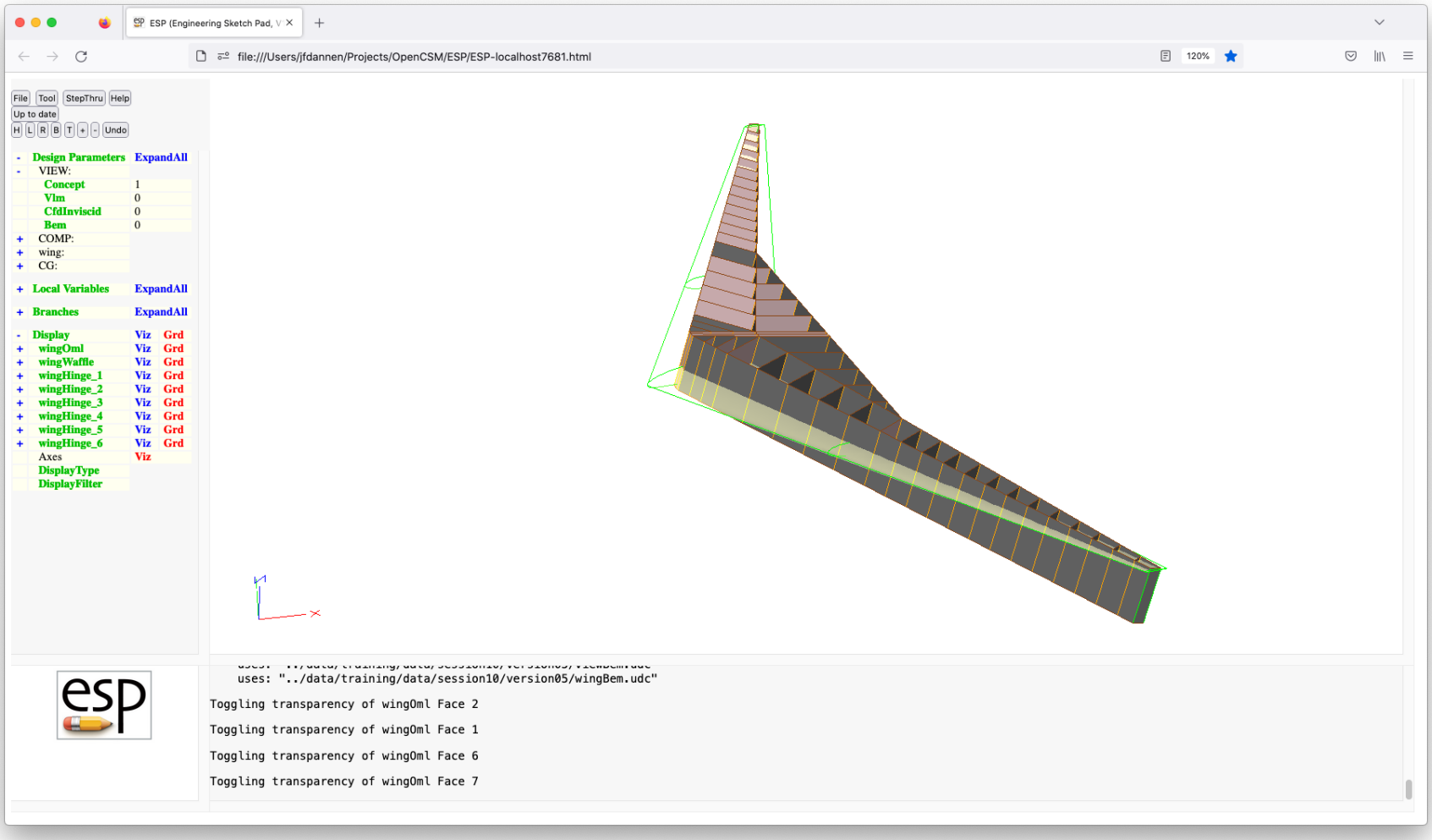
Build Up Config. in Multiple Versions — 2

Note: a `unittest` file should be created for every starred `.udc` file

10. add `fusePmtrs.udc`, `fuseCalc.udc`, and `fuseOml.udc*`
11. add `fuseIml.udc*`, `fuseWaffle.udc*`, and `fuseBem.udc*`
12. add `nacellePmtrs.udc`, `nacelleCalc.udc`, `nacelleOml.udc*`, `pylonPmtrs.udc`, `pylonCalc.udc`, and `pylonOml.udc*`
13. add `payloadPmtrs.udc` and `payload.udc*`
14. add `viewCfdViscous.udc*`
15. add `CAPS` Attributes to all `view*` files
16. add `viewCantilevel.udc*`, `viewSimpleSupport.udc*`, and `viewSkins.udc*`

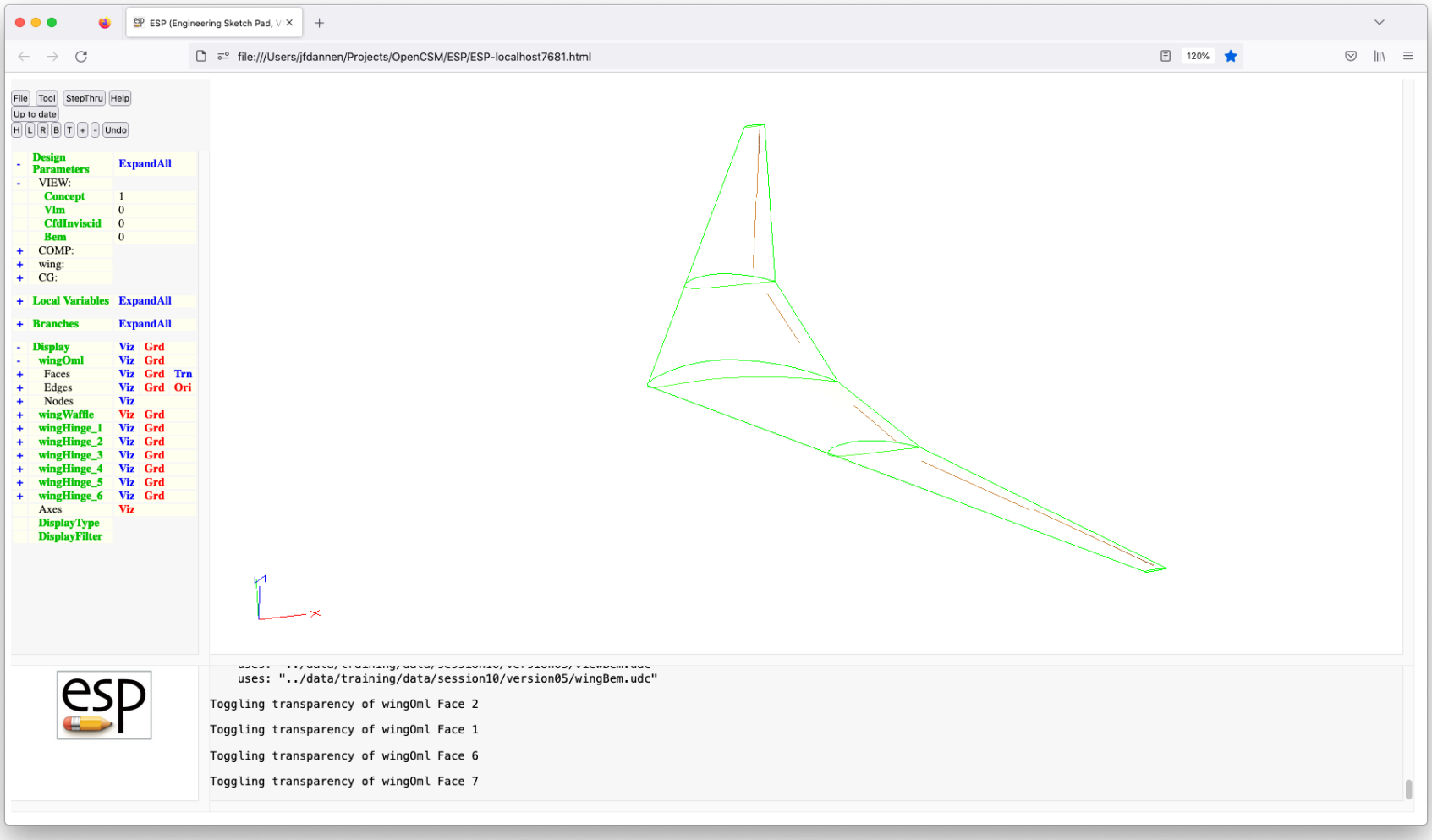
Example "views" for Version 5

`viewConcept` showing all Primitives



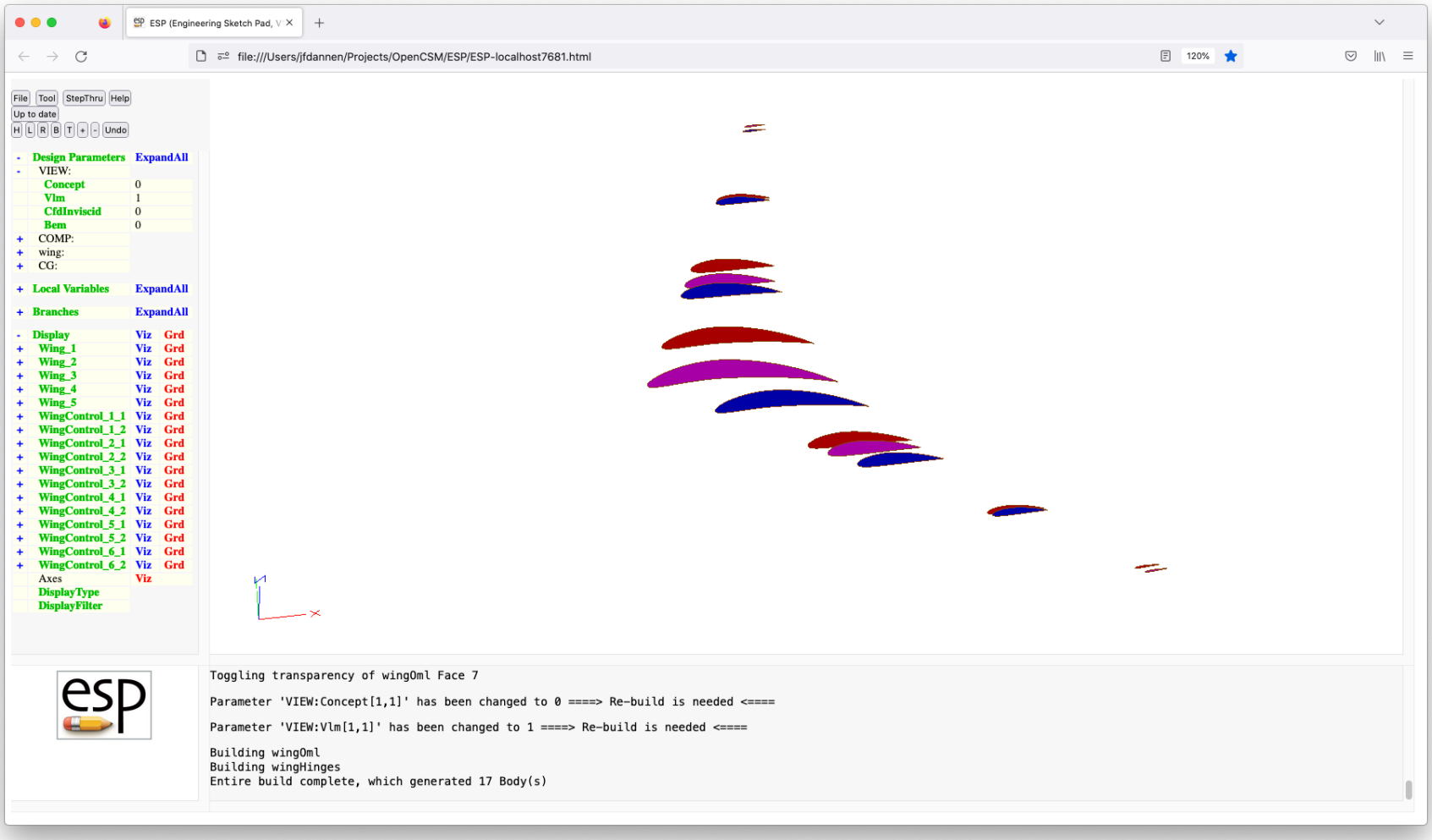
Example "views" for Version 5

`viewConcept` with waffle off and transparent OML



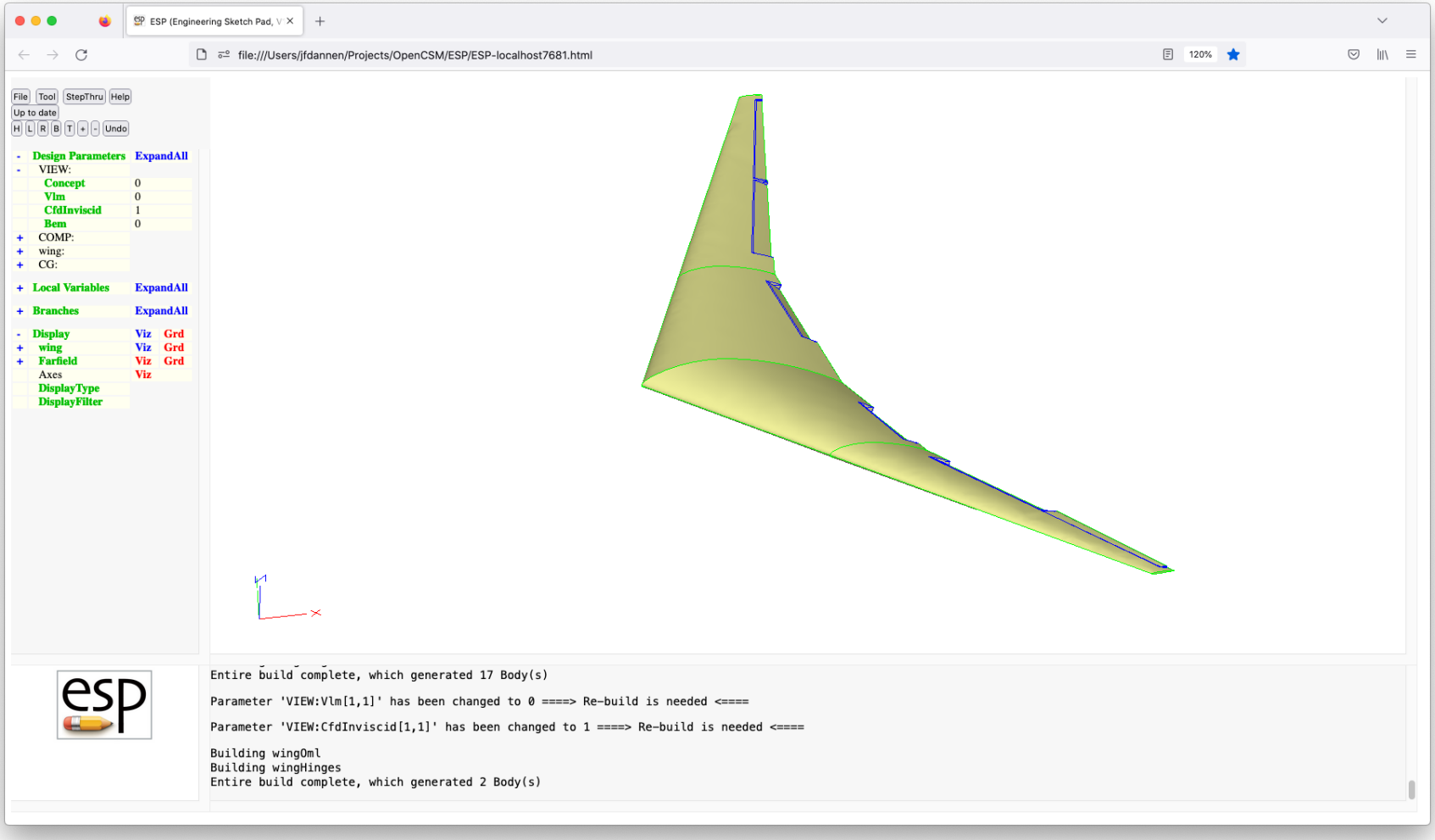
Example "views" for Version 5

viewVlm



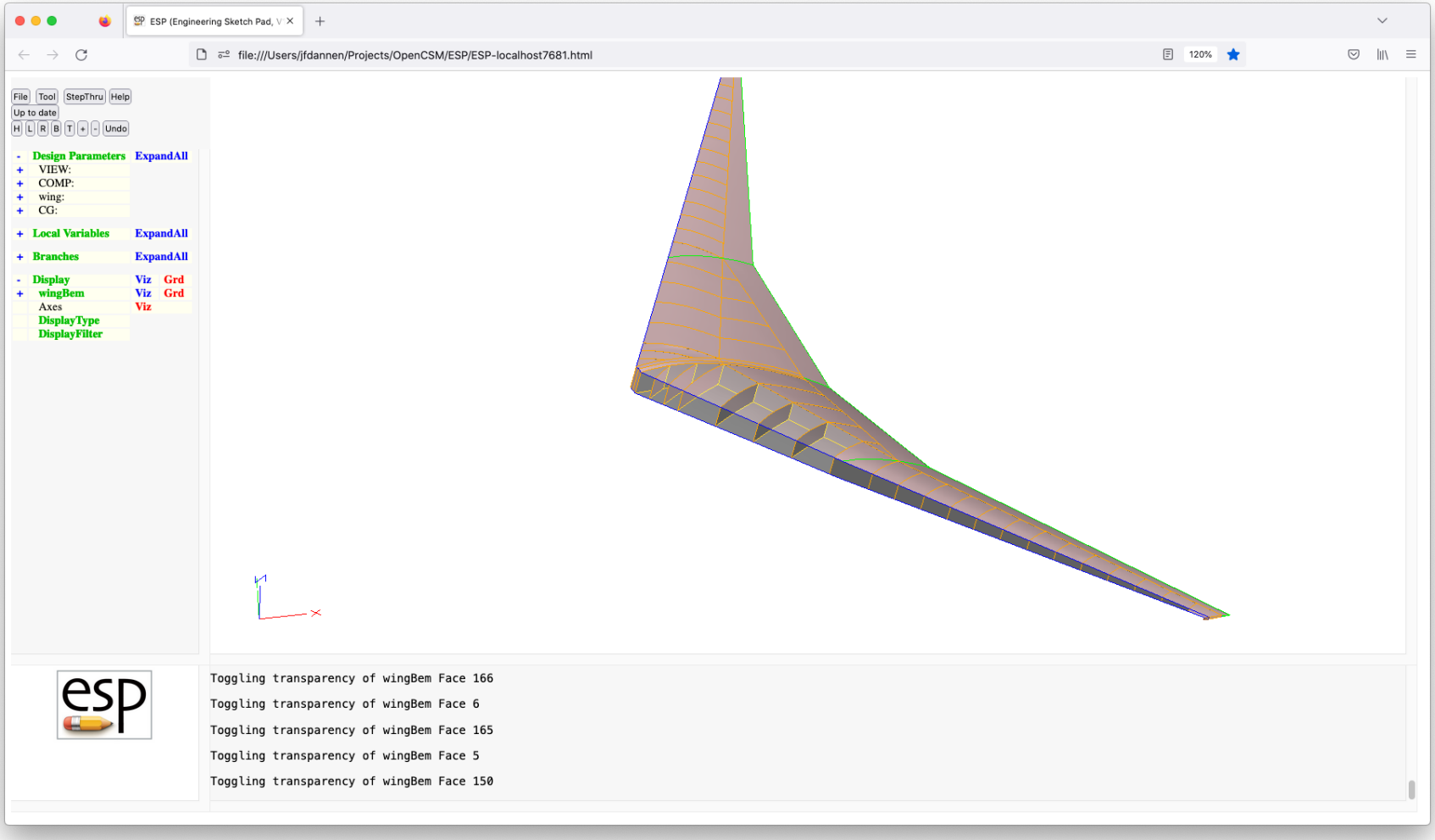
Example "views" for Version 5

viewCfdInviscid



Example "views" for Version 5

`viewBem` with some transparent panels



wingPmtrs.udc — 1

```
# .udc to define the DESPMTRs and CFGPMTRs for a wing
# written by John Dannenhoffer

INTERFACE . ALL

# wing Oml
DESPMTR   wing:area        4240    # area
DESPMTR   wing:aspect      9.00    # aspect ratio
DESPMTR   wing:taperi      0.48    # inboard  taper ratio
DESPMTR   wing:tapero      0.23    # outboard taper ratio
DESPMTR   wing:sweep       35.0    # leading edge sweep
DESPMTR   wing:dihedral     7.0    # dihedral
DESPMTR   wing:break       0.37    # inboard/outboard
DESPMTR   wing:alphar      -1.0    # setting angle   at root
DESPMTR   wing:thickr      0.10    # thickness ratio at root
DESPMTR   wing:camberr     0.08    # camber    ratio at root
DESPMTR   wing:alphab      -3.0    # setting angle   at break
DESPMTR   wing:thickb      0.15    # thickness ratio at break
DESPMTR   wing:camberb     0.04    # camber    ratio at break
DESPMTR   wing:alphat      -8.0    # setting angle   at tip
DESPMTR   wing:thickt      0.08    # thickness ratio at tip
DESPMTR   wing:cambert     0.01    # camber    ratio at tip
DESPMTR   wing:xroot       50.0    # xloc at root LE
DESPMTR   wing:zroot       -8.0    # zloc at root LE
```

wingPmtrs.udc — 2

```
# wing hinge lines
DIMENSION wing:hinge        6 9 1 # ymin                    ymax
#                           theta   x/c   y/span  z/t      x/c   y/span z/t    gap
DESPMTR   wing:hinge       "-10.0;  0.75; -0.98;  0.50;    0.75; -0.70; 0.50;  0.25;
                            +10.0;  0.75; -0.69;  0.00;    0.75; -0.43; 0.00;  0.25;
                            +15.0;  0.85; -0.33;  0.00;    0.90; -0.14; 0.00;  0.25;
                            +15.0;  0.90;  0.14;  0.00;    0.85;  0.33; 0.00;  0.25;
                            +10.0;  0.75;  0.43;  0.00;    0.75;  0.69; 0.00;  0.25;
                            +10.0;  0.75;  0.70;  0.50;    0.75;  0.98; 0.50;  0.25;

# wing structure
DESPMTR   wing:spar1       0.20    # fraction of chord for LE spar
DESPMTR   wing:spar2       0.70    # fraction of chord for TE spar
CFGPMTR   wing:nrib1          2    # number of internal ribs in region 1
CFGPMTR   wing:nrib2          4    # number of internal ribs in region 1
CFGPMTR   wing:nrib3         12    # number of internal ribs in region 1

DESPMTR   wing:waffleGap       1    # distance between fuselage and wing root rib

DESPMTR   wing:dxnom         2.0    # nominal .bdf element side length

END
```

wingCalc.udc

```
# .udc to calculate critial locations and dimensions for a wing
# written by John Dannenhoffer

INTERFACE . ALL

OUTPMTR   wing:mac
OUTPMTR   wing:span

SET       wing:span      sqrt(wing:aspect*wing:area)
SET       wing:yroot     0
SET       wing:ytip     -wing:span/2
SET       wing:xtip      wing:xroot-wing:ytip*tand(wing:sweep)
SET       wing:ztip      wing:zroot-wing:ytip*tand(wing:dihedral)
SET       wing:ybreak    wing:ytip*wing:break
SET       wing:xbreak    wing:xroot-wing:ybreak*tand(wing:sweep)
SET       wing:zbreak    wing:zroot-wing:ybreak*tand(wing:dihedral)
SET       wing:chordr    wing:area/((wing:yroot-wing:ybreak)*(wing:taperi+1)+(wing:
SET       wing:chordb    wing:chordr*wing:taperi
SET       wing:chordt    wing:chordb*wing:tapero
SET       wing:mac       sqrt(wing:area/wing:aspect)
SET       wing:sharpte   SHARP_TE

END
```

wingOml.udc — 1

```
# .udc to make the wingOml
# written by John Dannenhoffer

INTERFACE . ALL

# set a mark so that we can restore the stack back to
#    where it was when we started
MARK

# check to see if the Body already exits
RESTORE   wingOml

# if it does not exist, make it now
CATBEG    $name_not_found
   MESSAGE Building_wingOml

   # lay out left wing
   MARK
      # root
      UDPRIM    naca      thickness  wing:thickr   camber  wing:camberr   sharpte
      SCALE     wing:chordr
      ROTATEX   90  0  0
      ROTATEY   wing:alphar  0  0
      TRANSLATE wing:xroot   wing:yroot   wing:zroot
```

wingOml.udc — 2

```
   UDPRIM    naca      camber  wing:camberb   thickness  wing:thickb   sharpte
   SCALE     wing:chordb
   ROTATEX   90  0  0
   ROTATEY   wing:alphab  0  0
   TRANSLATE wing:xbreak       wing:ybreak    wing:zbreak

   # left tip
   UDPRIM    naca      thickness  wing:thickt   camber  wing:cambert   sharpte
   SCALE     wing:chordt
   ROTATEX   90  0  0
   ROTATEY   wing:alphat  0           0
   TRANSLATE wing:xtip    wing:ytip   wing:ztip
RULE
   ATTRIBUTE tagComp  $leftWing
SET       ruledBody @nbody

SELECT    FACE ruledBody  1
   ATTRIBUTE tagType  $root
SELECT    FACE ruledBody  2
   ATTRIBUTE tagType  $tip
   ATTRIBUTE tagIndex $1
SELECT    FACE ruledBody  3
   ATTRIBUTE tagType  $upper
```

wingOml.udc — 3

```
SELECT    FACE ruledBody  4
   ATTRIBUTE tagType  $upper
SELECT    FACE ruledBody  5
   ATTRIBUTE tagType  $lower
SELECT    FACE ruledBody  6
   ATTRIBUTE tagType  $lower
SELECT    EDGE ruledBody 3 ruledBody 5 1
   ATTRIBUTE tagType  $leadingEdge
SELECT    EDGE ruledBody 4 ruledBody 6 1
   ATTRIBUTE tagType  $leadingEdge
IFTHEN    wing:sharpte EQ 0
   SELECT    FACE ruledBody 7
      ATTRIBUTE tagType  $trailingEdge
   SELECT    FACE ruledBody 8
      ATTRIBUTE tagType  $trailingEdge
ELSE
   SELECT    EDGE  ruledBody 3 ruledBody 5 2
      ATTRIBUTE tagType  $trailingEdge
   SELECT    EDGE  ruledBody 4 ruledBody 6 2
      ATTRIBUTE tagType  $trailingEdge
ENDIF
```

wingOml.udc — 4

```
# right wing too
STORE     LeftWing 0 1
RESTORE   LeftWing
   ATTRIBUTE tagComp $riteWing
SELECT    FACE  $tagType $tip
   ATTRIBUTE tagIndex $2
SELECT    EDGE  $tagType $leadingEdge
IFTHEN    @iedge GT 0
   SELECT EDGE  $tagType $leadingEdge
      ATTRIBUTE tagComp $riteWing
ENDIF
IFTHEN    wing:sharpte EQ 1
   SELECT    EDGE  $tagType $trailingEdge
   IFTHEN    @iedge GT 0
      SELECT EDGE  $tagType $trailingEdge
         ATTRIBUTE tagComp $riteWing
   ENDIF
ENDIF
MIRROR    0   1  0

# join into single wing
JOIN
```

wingOml.udc — 5

```
   # attribute the root
   SELECT    EDGE  ruledBody 3 ruledBody 3 1
      ATTRIBUTE tagType  $root
   SELECT    EDGE  ruledBody 5 ruledBody 5 1
   ATTRIBUTE tagType  $root

   # store the final Body
   STORE     wingOml
CATEND

# make sure that we did not leave any new Bodys on the stack
STORE     ..

END
```

wingHinges.udc — 1

```
# .udc to make the wingHinges
# written by John Dannenhoffer

INTERFACE . ALL

# set a mark so that we can restore the stack back to
#    where it was when we started
MARK

# skip this if controls are off
IFTHEN    COMP:controls EQ 0
   THROW  -7681
ENDIF

# check to see if the Body already exists
RESTORE   wingHinge 1

# if they do not exist, make them now
CATBEG    $name_not_found
   MESSAGE Building_wingHinges

   # make sure the wingOml exists
   UDPRIM    $/wingOml
```

wingHinges.udc — 2

```
PATBEG    ihinge  wing:hinge.nrow
   SET       y_ibd   wing:hinge[ihinge,3]*(-wing:ytip)
   BOX       -1000  y_ibd  -1000  2000  0  2000
   RESTORE   wingOml
   INTERSECT
   SET       x_ibd @xmin+wing:hinge[ihinge,2]*(@xmax-@xmin)
   STORE     .
   BOX       x_ibd  y_ibd  -1000  0  0  2000
   RESTORE   wingOml
   INTERSECT
   SET       z_ibd   @zmin+wing:hinge[ihinge,4]*(@zmax-@zmin)
   STORE     .

   SET       y_obd   wing:hinge[ihinge,6]*(-wing:ytip)
   BOX       -1000  y_obd  -1000  2000  0  2000
   RESTORE   wingOml
   INTERSECT
   SET       x_obd   @xmin+wing:hinge[ihinge,5]*(@xmax-@xmin)
   STORE     .
   BOX       x_obd  y_obd  -1000  0  0  2000
   RESTORE   wingOml
   INTERSECT
   SET       z_obd   @zmin+wing:hinge[ihinge,7]*(@zmax-@zmin)
   STORE     .
```

wingHinges.udc — 3

```
      SKBEG       x_ibd  y_ibd  z_ibd
         LINSEG x_obd  y_obd  z_obd
      SKEND
      SELECT    EDGE 1
         ATTRIBUTE tagComp  $wing
         ATTRIBUTE tagType  $hinge
         ATTRIBUTE tagIndex !val2str(wing:hinge[ihinge,9],0)
         ATTRIBUTE deflect   wing:hinge[ihinge,1]
         ATTRIBUTE xoverc1   wing:hinge[ihinge,2]
         ATTRIBUTE xoverc2   wing:hinge[ihinge,5]
         ATTRIBUTE gap       wing:hinge[ihinge,8]
         ATTRIBUTE compIndex !val2str(ihinge,0)
      SELECT BODY
         ATTRIBUTE _name     $wingHinge_+ihinge

      # store the final Body
      STORE   wingHinge ihinge
   PATEND
CATEND
```

wingHinges.udc — 4

```
CATBEG    -7681
CATEND

# make sure that we did not leave any new Bodys on the stack
STORE     ..

END
```

wingWaffle.udc — 1

```
# .udc to make the wingWaffle
# written by John Dannenhoffer

INTERFACE . ALL

# set a mark so that we can restore the stack back to
#    where it was when we started
MARK

# check to see if the Body already exits
RESTORE   wingWaffle

# if it does not exist, make it now
CATBEG    $name_not_found
   MESSAGE Building_wingWaffle

   # make sure the wingOml exists
   UDPRIM    $/wingOml

   # outline of the waffle
   SET         yA             0
   SET         xA             wing:xroot
   SET         yB             -wing:ytip
```

wingWaffle.udc — 2

```
SET         xB              wing:xtip
SET         yC              0
SET         xC              wing:xroot+wing:chordr
SET         yD              -wing:ybreak
SET         xD              wing:xbreak+wing:chordb
SET         yE              -wing:ytip
SET         xE              wing:xtip+wing:chordt

# get required depth of the waffle
RESTORE     wingOml
SET         zmin            @zmin-0.1
SET         zmax            @zmax+0.1
STORE       .

# make the waffle
UDPARG      waffle          depth     zmax-zmin
UDPRIM      waffle          filename <<

# construction lines for wing outline
CPOINT A    AT  xA  yA
CPOINT B    AT  xB  yB
CPOINT C    AT  xC  yC
CPOINT D    AT  xD  yD
CPOINT E    AT  xE  yE
```

wingWaffle.udc — 3

```
CLINE  AB      A   B
CLINE  CD      C   D
CLINE  DE      D   E
CLINE  AC      A   C
CLINE  BE      B   E

# construction lines for fuselage side and wing break
CPOINT F   ON  AB  YLOC  y@D
CPOINT K   ON  AB  YLOC  wing:waffleGap
CPOINT L   ON  CD  YLOC  wing:waffleGap

CLINE  FD      F   D

# construction lines for spars
CPOINT G   ON  FD   FRAC  wing:spar1
CPOINT H   ON  BE   FRAC  wing:spar1
CPOINT I   ON  FD   FRAC  wing:spar2
CPOINT J   ON  BE   FRAC  wing:spar2

CLINE  GH      G   H
CLINE  IJ      I   J
CLINE  KL      K   L
```

wingWaffle.udc — 4

```
# spars
POINT  M   ON  GH  XSECT  KL
LINE   MH      M   H  tagType=spar  tagIndex=1

POINT  N   ON  IJ  XSECT  KL
LINE   NJ      N   J  tagType=spar  tagIndex=2

POINT  O   ON  KL  XLOC   x@I
LINE   OI      O   I  tagType=spar  tagIndex=3

# fuselage wing box
POINT  MM  AT  x@M 0
LINE   .       M   MM tagType=fusespar tagIndex=1

POINT  X   AT  x@N 0
LINE   .       N   X  tagType=fusespar tagIndex=2

POINT  OO  at  x@O 0
LINE   .       O   OO tagType=fusespar tagIndex=3

# rib
LINE MN M N
LINE NO N O
```

wingWaffle.udc — 5

```
# wing root
LINE   .       MM  OO tagType=root

# wing tip
LINE   HJ      H   J  tagType=tip

# ribs in region 1
PATBEG  iIi  wing:nrib1
   POINT  X   AT  x@M+(x@N-x@M)*iIi/(wing:nrib1+1) y@M+(y@N-y@M)*iIi/(wing:nrib1
   POINT  Y   ON  MH  PERP   X
   LINE   .       X   Y   tagType=rib   tagIndex=!val2str(iIi,0)
PATEND

# rib from point N
CPOINT X   AT      x@N    y@N
POINT  Y   ON  MH  PERP   X
LINE   .       X   Y      tagType=rib   tagIndex=!val2str(wing:nrib1+1,0)

# ribs in region 2
PATBEG  iIi  wing:nrib2
   POINT  X  AT x@N+(x@I-x@N)*iIi/(wing:nrib2+1)  y@N+(y@I-y@N)*iIi/(wing:nrib2+
   POINT  Y   ON  MH  PERP   X
   LINE   .       Y   X      tagType=rib   tagIndex=!val2str(wing:nrib1+2+iIi,0)
   POINT  Z   ON  OI  PERP   X
   LINE           X   Z      tagType=rib   tagIndex=!val2str(wing:nrib1+2+iIi,0)
```

wingWaffle.udc — 6

```
# rib from point I
CPOINT X   AT      x@I    y@I
POINT  Y   ON  MH  PERP   X
LINE   .       X   Y      tagType=rib   tagIndex=!val2str(wing:nrib1+wing:nrib2+

# ribs in region 3
PATBEG  iIi  wing:nrib3
   POINT  X  AT x@I+(x@J-x@I)*iIi/(wing:nrib3+1)  y@I+(y@J-y@I)*iIi/(wing:nrib3+
   POINT  Y   ON  MH  PERP   X
   LINE   .       Y   X   tagType=rib   tagIndex=!val2str(wing:nrib1+wing:nrib2+
PATEND
>>

# move down to be coincident with wingOml
TRANSLATE  0  0  zmin

# attribute the rite wing Faces
SELECT    FACE
   ATTRIBUTE tagComp $riteWing

# make a copy for the left wing
RESTORE   .
MIRROR    0  1  0  0
```

wingWaffle.udc — 7

```
   # re-attribute the left wing Faces
   SELECT    FACE
      ATTRIBUTE tagComp $leftWing

   # make a single waffle
   JOIN

   # get the locations of the wing spars through the fuselage
   SELECT    FACE          $tagType  $fusespar  $tagIndex  $1
   SET       wing:xspar1   @xcg
   SELECT    FACE          $tagType  $fusespar  $tagIndex  $2
   SET       wing:xspar2   @xcg
   SELECT    FACE          $tagType  $fusespar  $tagIndex  $3
   SET       wing:xspar3   @xcg

   # store the final Body
   STORE     wingWaffle
CATEND

# make sure that we did not leave any new Bodys on the stack
STORE     ..

END
```

viewConcept.udc

```
# .udc to make the Concept view
# written by John Dannenhoffer

INTERFACE . ALL

# make sure we have the necessary Bodys
IFTHEN    COMP:wing NE 0
    UDPRIM    $/wingOml
    UDPRIM    $/wingWaffle
    UDPRIM    $/wingHinges
ENDIF

# now that we have all the Bodys, show them
IFTHEN    COMP:wing NE 0
   RESTORE    wingOml
      ATTRIBUTE _name $wingOml
   RESTORE    wingWaffle
      ATTRIBUTE _name $wingWaffle
   PATBEG     ihinge  wing:hinge.nrow*COMP:controls
      RESTORE wingHinge ihinge
   PATEND
ENDIF

END
```

viewOml.udc

```
# .udc to make the Oml view
# written by John Dannenhoffer

INTERFACE . ALL

# make sure we have the necessary Bodys
IFTHEN    COMP:wing NE 0
    UDPRIM    $/wingOml
    UDPRIM    $/wingHinges
ENDIF

# get the wing
IFTHEN    COMP:wing NE 0
   RESTORE   wingOml
      ATTRIBUTE _name $wing
ENDIF

END
```

viewCfdInviscid.udc — 1

```
# .udc to make the CfdInvisicd view
# written by John Dannenhoffer

INTERFACE . ALL

# get the Oml first
UDPRIM    $/viewOml

DIMENSION  xflap  1  4
DIMENSION  yflap  1  4

# since we are going to restore the various Bodys (below)
#    clear the stack now
STORE     ...

# add the control surfaces to the wing
IFTHEN    COMP:wing NE 0

   # set the at-parameters for the wing
   RESTORE   wingOml
```

wingCfdInviscid.udc — 2

```
PATBEG ihinge wing:hinge.nrow*COMP:controls

   # aileron and outboard flap
   IFTHEN wing:hinge[ihinge,3] LT -wing:break OR wing:hinge[ihinge,3] GT +wing:b
      SET  s (abs(wing:hinge[ihinge,6])-wing:break)/(1-wing:break)
      SET c2 wing:chordb*(1-s)+wing:chordt*s

      SET xflap[1] @xmax+1
      SET yflap[1] wing:hinge[ihinge,6]*wing:span/2

      SET xflap[2] wing:xbreak*(1-s)+wing:xtip*s+c2*wing:hinge[ihinge,5]
      SET yflap[2] yflap[1]

      SET  s (abs(wing:hinge[ihinge,3])-wing:break)/(1-wing:break)
      SET c3 wing:chordb*(1-s)+wing:chordt*s

      SET xflap[3] wing:xbreak*(1-s)+wing:xtip*s+c3*wing:hinge[ihinge,2]
      SET yflap[3] wing:hinge[ihinge,3]*wing:span/2

      SET xflap[4] xflap[1]
      SET yflap[4] yflap[3]
```

wingCfdInviscid.udc — 3

```
# inboard flaps
ELSE
   SET  s abs(wing:hinge[ihinge,6])/wing:break
   SET c2 wing:chordr*(1-s)+wing:chordb*s

   SET xflap[1] @xmax+1
   SET yflap[1] wing:hinge[ihinge,6]*wing:span/2

   SET xflap[2] wing:xroot*(1-s)+wing:xbreak*s+c2*wing:hinge[ihinge,5]
   SET yflap[2] yflap[1]

   SET  s abs(wing:hinge[ihinge,3])/wing:break
   SET c3 wing:chordr*(1-s)+wing:chordb*s

   SET xflap[3] wing:xroot*(1-s)+wing:xbreak*s+c3*wing:hinge[ihinge,2]
   SET yflap[3] wing:hinge[ihinge,3]*wing:span/2

   SET xflap[4] xflap[1]
   SET yflap[4] yflap[3]
ENDIF

# generate the flap
UDPARG    $$/flapz xflap xflap
UDPARG    $$/flapz yflap yflap
UDPRIM    $$/flapz gap wing:hinge[ihinge,8] theta wing:hinge[ihinge,1]
```

wingCfdInviscid.udc — 4

```
      IFTHEN    yflap.min GT 0
         UDPRIM    editAttr  filename  <<
            FACE    HAS  *=*
            ANDNOT  HAS  tagComp=*
            SET          tagComp=riteWing
            SET          tagType=filler
         >>
      ELSE
         UDPRIM    editAttr  filename  <<
            FACE    HAS  *=*
            ANDNOT  HAS  tagComp=*
            SET          tagComp=leftWing
            SET          tagType=filler
         >>
      ENDIF
   PATEND

   SELECT    BODY
      ATTRIBUTE _name $wing
ENDIF
```

wingCfdInviscid.udc — 5

```
# get the extrema
SET       nbody    @stack.size
PATBEG    ibody    nbody
   STORE  tempBody ibody
PATEND

SET       xmin     +1e20
SET       xmax     -1e20
SET       ymin     +1e20
SET       ymax     -1e20
SET       zmin     +1e20
SET       zmax     -1e20

PATBEG    ibody    nbody
   RESTORE   tempBody  nbody+1-ibody
   SET    xmin     min(xmin,@xmin)
   SET    xmax     max(xmax,@xmax)
   SET    ymin     min(ymin,@ymin)
   SET    ymax     max(ymax,@ymax)
   SET    zmin     min(zmin,@zmin)
   SET    zmax     max(zmax,@zmax)
PATEND
```

wingCfdInviscid.udc — 6

```
# build the farfield
SET       size   20*max(xmax-xmin,ymax-ymin)
BOX       (xmin+xmax-size)/2   (ymin+ymax-size)/2   (zmin+zmax-size)/2 \
          size                 size                 size
   ATTRIBUTE _name              $Farfield

END
```

viewVlm.udc

```
# .udc to make the Vlm view
# written by John Dannenhoffer

INTERFACE . ALL

# make sure we have the necessary Bodys
IFTHEN    COMP:wing NE 0
   UDPRIM    $/wingVlm
ENDIF

# now that we have all the Bodys, show and Attribute them
IFTHEN    @stack[1] EQ -1
   SET    prevStack 0
ELSE
   SET    prevStack @stack.size
ENDIF

# wing
IFTHEN    COMP:wing NE 0
   RESTORE     wingVlm
ENDIF

END
```

viewBem.udc

```
# .udc to make the Bem
# written by John Dannenhoffer

INTERFACE . ALL

# make sure we have the wingBem
IFTHEN    COMP:wing   NE 0
   UDPRIM    $/wingBem
ENDIF

IFTHEN    COMP:wing   NE 0
   RESTORE   wingBem
      ATTRIBUTE _name $wingBem
ENDIF

END
```

transport_init.udc — 1

```
# .udc to set up DESPMTRs, CFGPMTRs, and critical locations and dimensions
# written by John Dannenhoffer

INTERFACE . ALL

# global tolerance
set EPS06 1.0e-6

# make a list of the components
CFGPMTR   COMP:wing     1

# controls must be either 0=off or 1=on
CFGPMTR   COMP:controls 1
IFTHEN    COMP:controls NE 0  AND  COMP:controls NE 1
   MESSAGE  COMP:controls_must_be_0_or_1
   THROW    -999
ENDIF

# define the DESPMTRs and CFGPMTRs
UDPRIM    $/wingPmtrs
```

transport_init.udc — 2

```
# put sharp trailing edges on all aero surfaces
SET       SHARP_TE   1

# compute critical locations / dimensions
UDPRIM    $/wingCalc

# CG location used to drive design parametres, not the actual CG
DIMENSION CG:ref  3 1
DESPMTR   CG:ref  "90; 0; 0"

END
```

transport.csm — 1

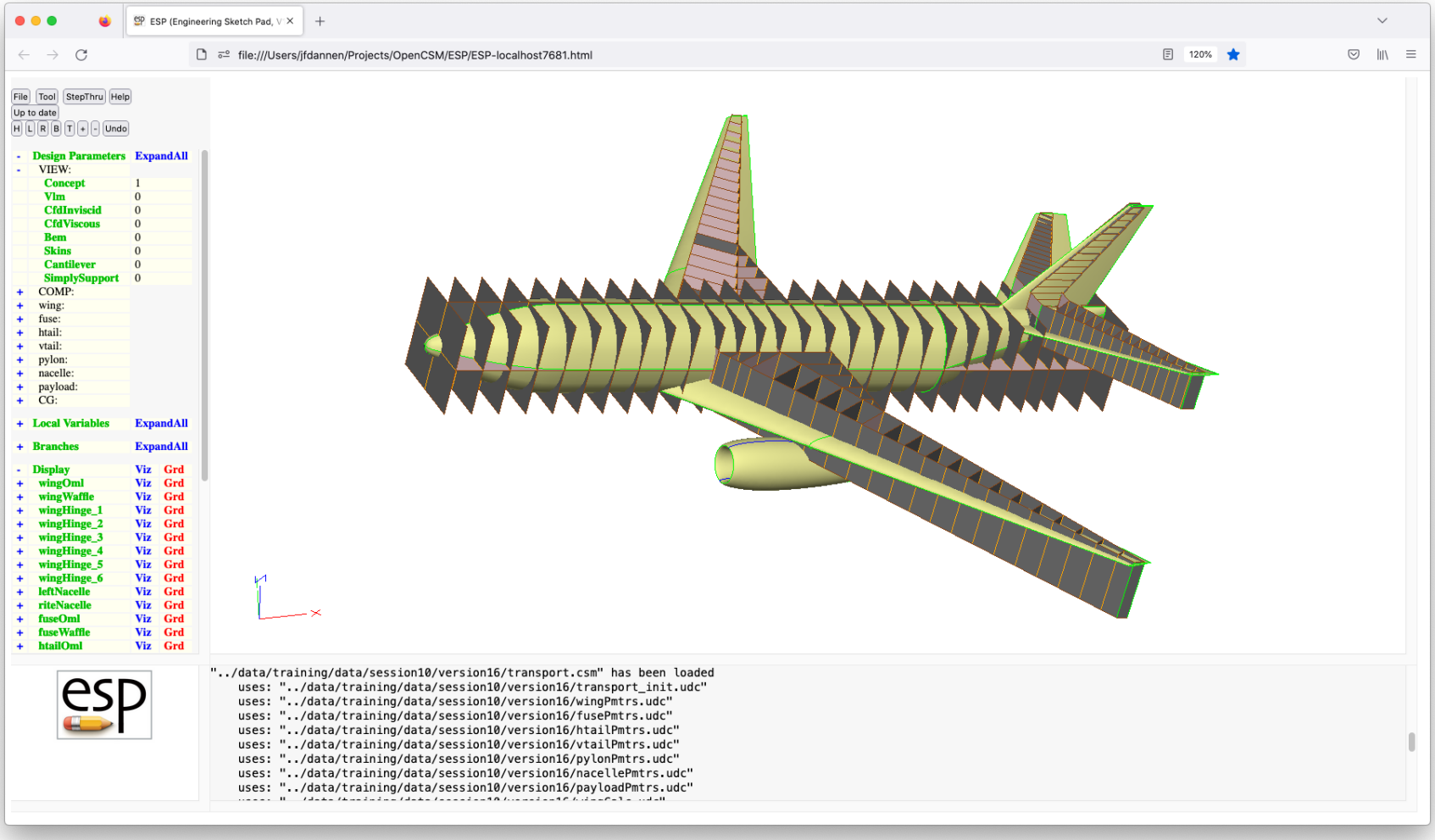
```
# transport
# written by John Dannenhoffer

# define the views
CFGPMTR   VIEW:Concept          1
CFGPMTR   VIEW:Vlm              0

CFGPMTR   VIEW:CfdInviscid      0

CFGPMTR   VIEW:Bem              0

UDPRIM    $/transport_init

IFTHEN    VIEW:Concept      NE  0
   UDPRIM    $/viewConcept
ENDIF
```

wingOml.udc — 2

```
IFTHEN    VIEW:Vlm           NE  0
   UDPRIM    $/viewVlm
ENDIF

IFTHEN    VIEW:CfdInviscid   NE  0
   UDPRIM    $/viewCfdInviscid
ENDIF

IFTHEN    VIEW:Bem           NE  0
   UDPRIM    $/viewBem
ENDIF

END
```

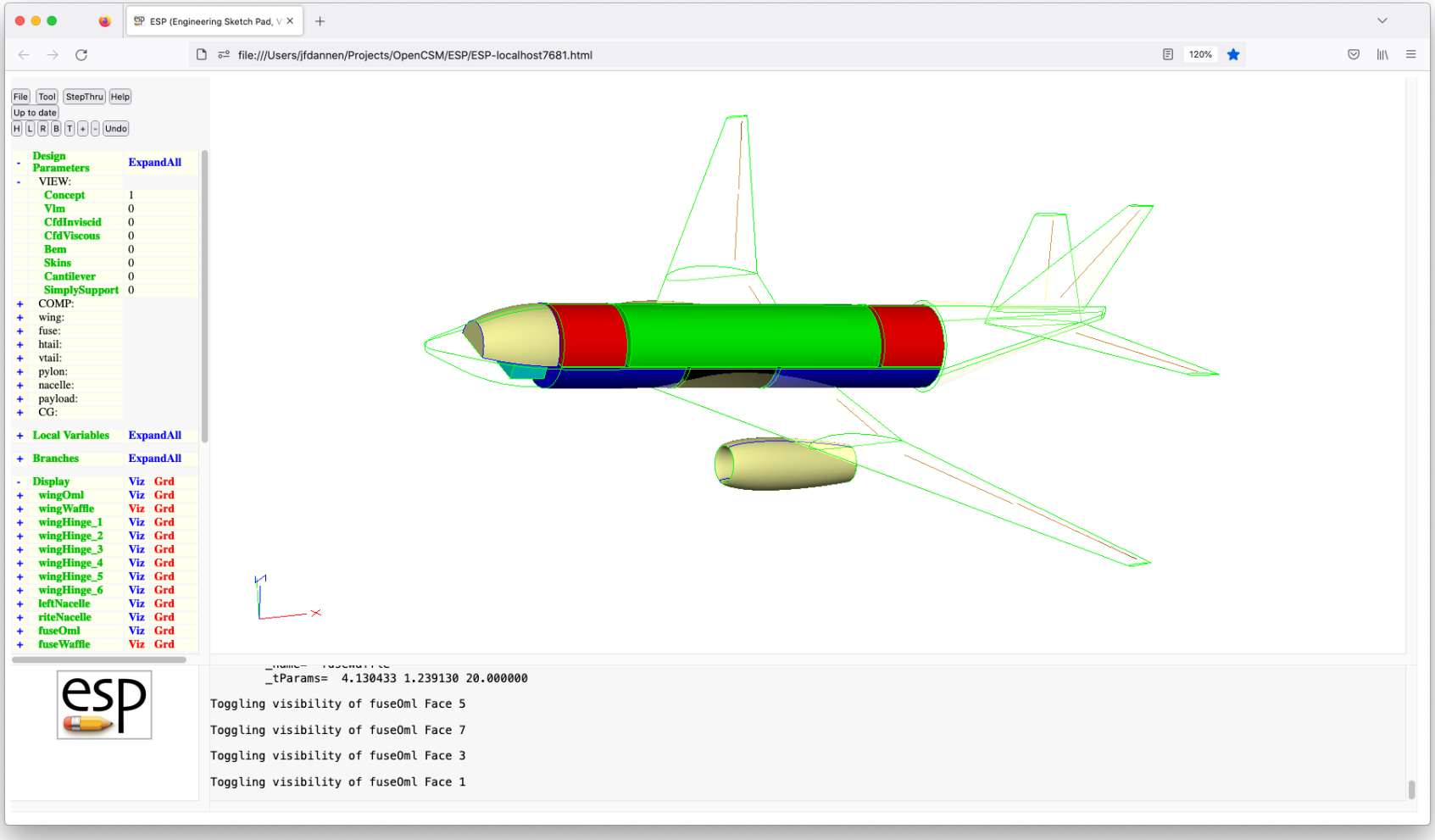
Example "views" for Version 16

`viewConcept` showing all Primitives



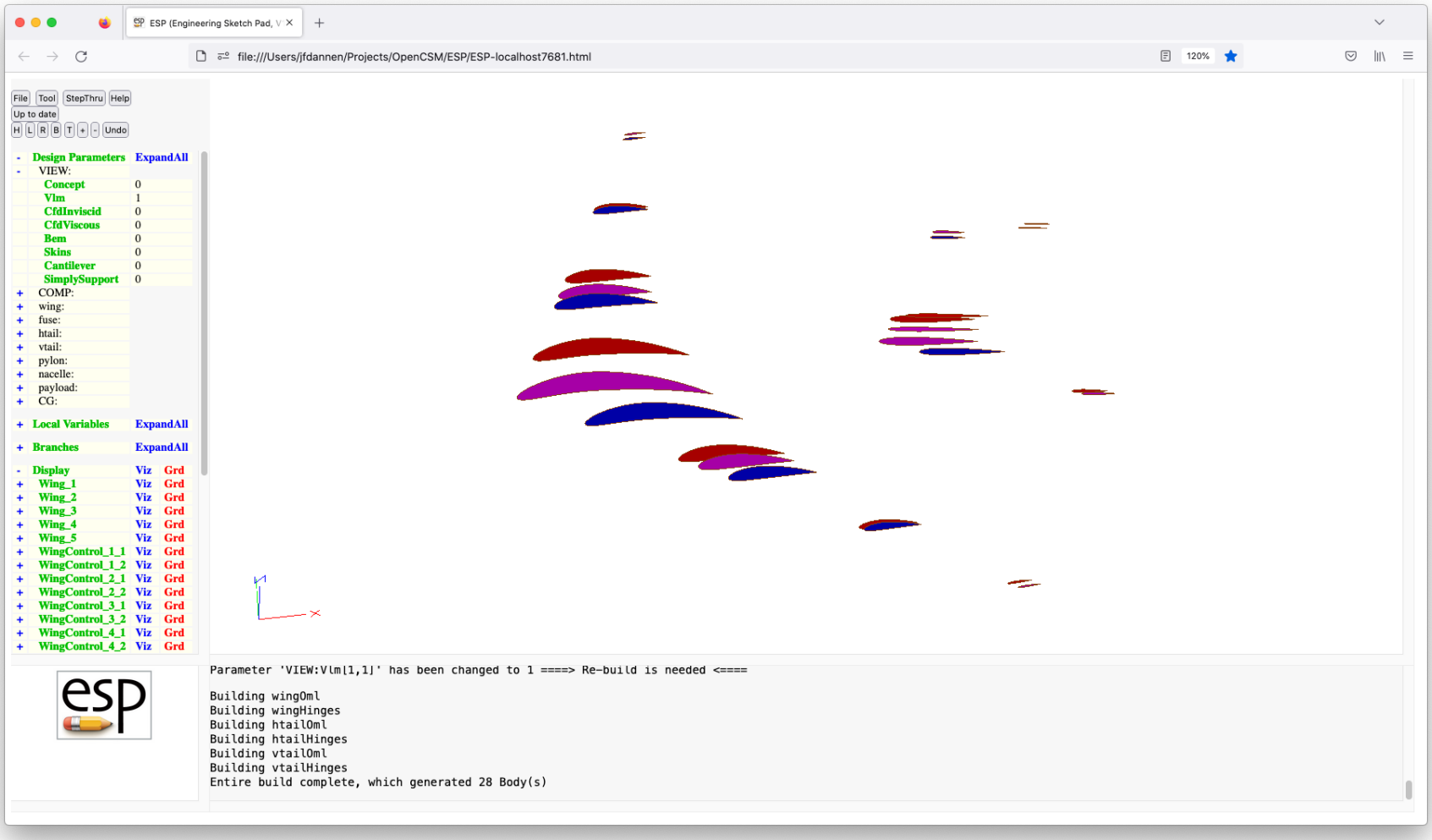
Example "views" for Version 16

`viewConcept` with waffle off and transparent OML



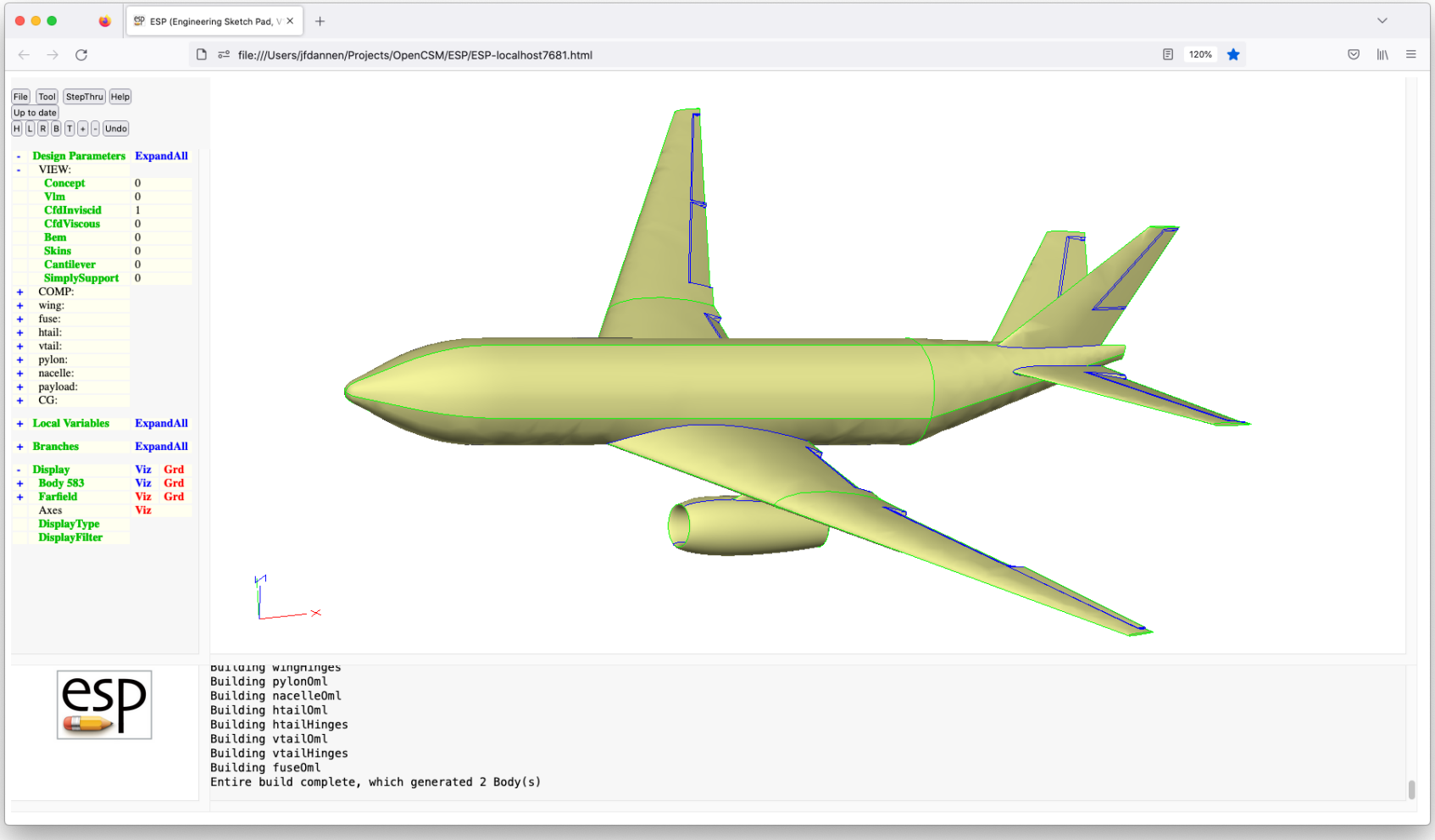
Example "views" for Version 16

viewVlm



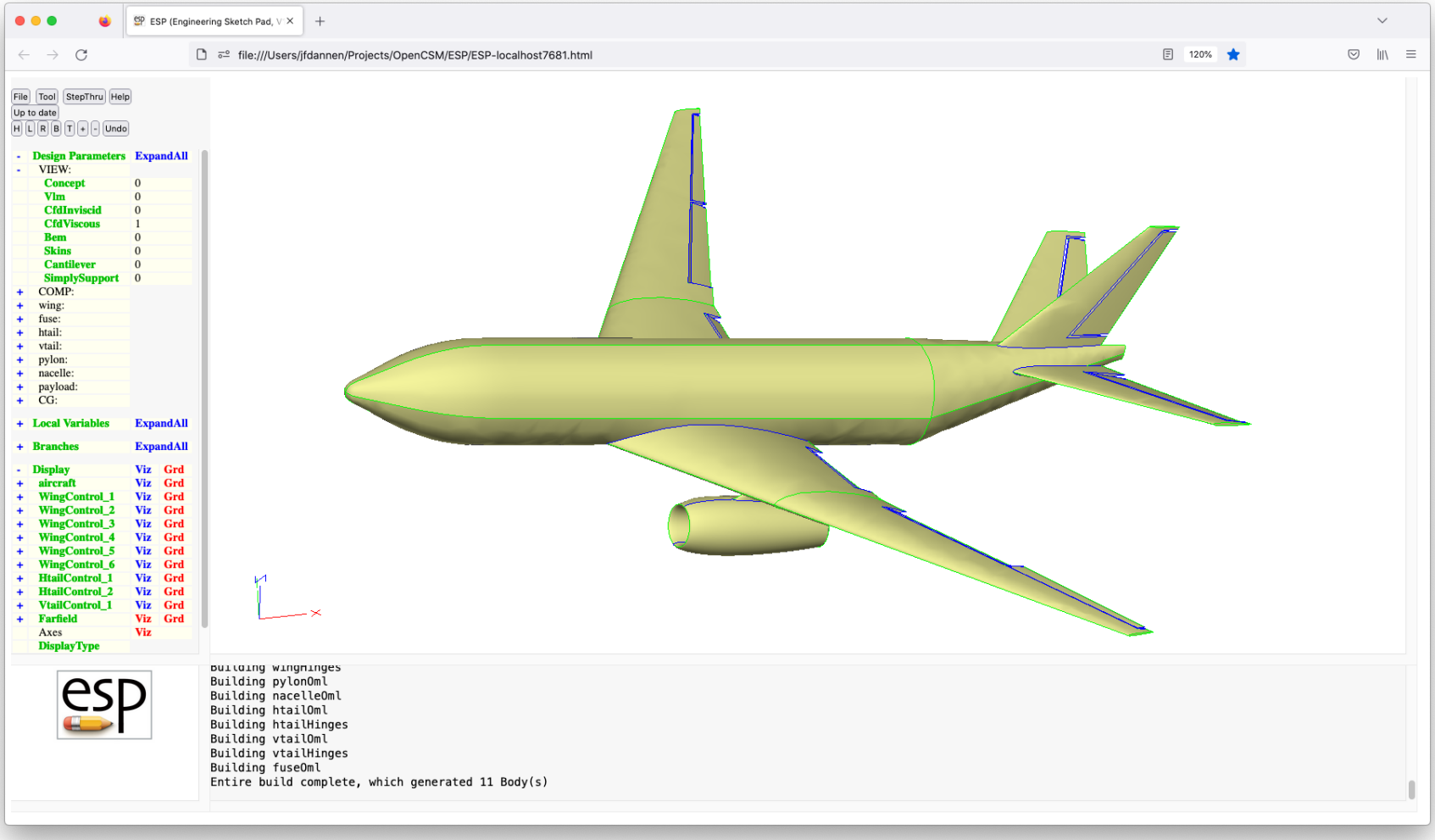
Example "views" for Version 16

viewCfdInviscid



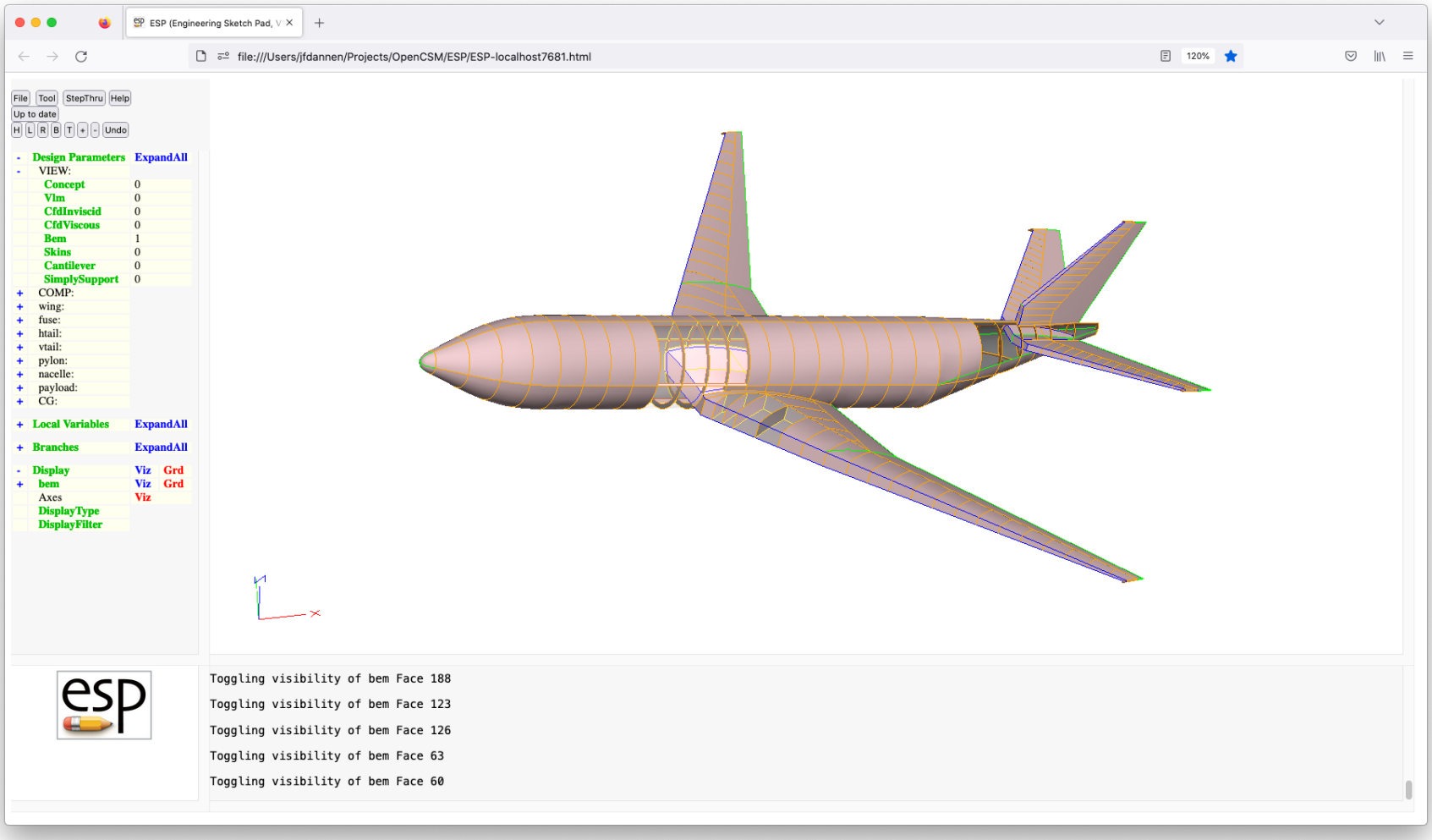
Example "views" for Version 16

viewCfdViscous



Example "views" for Version 16

`viewBem` with some transparent panels



Next Steps

- Explore this model
  - track the changes of any file over time (version)
  - examine how components interact with each other
    - fuselage length depends on tail placement
    - fuselage bulkheads depend on spars in wings and tail
    - nacelle and pylon depend on wing parametes
    - ...

Summary

- ESP is a powerful geometry-generating system that was designed for the analysis of complex configurations
  - supports multiple linked models
  - supports persistent attribution
  - provides sensitivities
  - can easily be coupled with other systems
- For CAPS, a set of "views" were defined; but these are only an example
- Each organization will want to develop a set of rules and conventions that are consistent with the rest of the organization's design systems

Final Thoughts

- `ESP` is freely available for download from `acdl.mit.edu/ESP`
- Based upon user requests, new and improved features are added continually
- Send bug reports to `jfdannen@syr.edu`
- Also send success stories to `jfdannen@syr.edu`

- Thank you for attending; send comments about the course to `jfdannen@syr.edu`