Masstran Analysis Interface Module (AIM) Manual

Marshall Galbraith
MIT ACDL

December 8, 2022

0.1 Introduction

0.1.1 Masstran AIM Overview

A module in the Computational Aircraft Prototype Syntheses (CAPS) has been developed to compute mass properties using attributions for finite element structural solvers.

An outline of the AIM's inputs, outputs and attributes are provided in AIM Inputs and AIM Outputs and Masstran AIM attributes, respectively.

Details on the use of units are outlined in AIM Units.

The mass properties are computed via the formulas:

$$
\begin{aligned}
m &= \sum_i m_i \\
x_{cg} &= \frac{1}{m}\sum_i m_i x_i \\
y_{cg} &= \frac{1}{m}\sum_i m_i y_i \\
z_{cg} &= \frac{1}{m}\sum_i m_i z_i \\
\left(I_{xx}\right)_{cg} &= \sum_i m_i\left(y_i^2+z_i^2\right) - m\left(y_{cg}^2+z_{cg}^2\right) \\
\left(I_{yy}\right)_{cg} &= \sum_i m_i\left(x_i^2+z_i^2\right) - m\left(x_{cg}^2+z_{cg}^2\right) \\
\left(I_{zz}\right)_{cg} &= \sum_i m_i\left(x_i^2+y_i^2\right) - m\left(x_{cg}^2+y_{cg}^2\right) \\
\left(I_{xy}\right)_{cg} &= \sum_i m_i\left(x_i y_i\right) - m\left(x_{cg} y_{cg}\right) \\
\left(I_{xz}\right)_{cg} &= \sum_i m_i\left(x_i z_i\right) - m\left(x_{cg} z_{cg}\right) \\
\left(I_{yz}\right)_{cg} &= \sum_i m_i\left(y_i z_i\right) - m\left(y_{cg} z_{cg}\right),
\end{aligned}
$$

where i represents an element index in the mesh, and the mass m_i is computed from the density, thickness, and area of the element.

The moment of inertias are accessible individually, in vector form as

$$\vec{I} = \begin{bmatrix} I_{xx} & I_{yy} & I_{zz} & I_{xy} & I_{xz} & I_{yz} \end{bmatrix},$$

as lower/upper triangular form

$$\vec{I}_{lower} = \begin{bmatrix} I_{xx} & -I_{xy} & I_{yy} & -I_{xz} & -I_{yz} & I_{zz} \end{bmatrix},$$

$$\vec{I}_{upper} = \begin{bmatrix} I_{xx} & -I_{xy} & -I_{xz} & I_{yy} & -I_{yz} & I_{zz} \end{bmatrix},$$

or in full tensor form as

$$\bar{\bar{I}} = \begin{bmatrix} I_{xx} & -I_{xy} & -I_{xz} \\ -I_{xy} & I_{yy} & -I_{yz} \\ -I_{xz} & -I_{yz} & I_{zz} \end{bmatrix}.$$

0.1.2 Examples

An example problem using the Masstran AIM may be found at Masstran AIM Basic Example.

0.2 Masstran AIM attributes

The following list of attributes are required for the MYSTRAN AIM inside the geometry input.

- **capsAIM** This attribute is a CAPS requirement to indicate the analysis the geometry representation supports.
- **capsGroup** This is a name assigned to any geometric body. This body could be a solid, surface, face, wire, edge or node. Recall that a string in ESP starts with a $. For example, attribute `capsGroup $Wing`.
- **capsIgnore** It is possible that there is a geometric body (or entity) that you do not want the Masstran AIM to pay attention to when creating a finite element model. The capsIgnore attribute allows a body (or entity) to be in the geometry and ignored by the AIM. For example, because of limitations in OpenCASCADE a situation where two edges are overlapping may occur; capsIgnore allows the user to only pay attention to one of the overlapping edges.

0.3 AIM Units

A unit system may be optionally specified during AIM instance initiation. If a unit system is provided, all AIM input values which have associated units must be specified as well. If no unit system is used, AIM inputs, which otherwise would require units, will be assumed unit consistent. A unit system may be specified via a JSON string dictionary for example: unitSys = "{"mass": "kg", "length": "m"}"

0.3.1 JSON String Dictionary

The key arguments of the dictionary are described in the following:

- **mass = "None"**
  Mass units - e.g. "kilogram", "k", "slug", ...
- **length = "None"**
  Length units - e.g. "meter", "m", "inch", "in", "mile", ...

0.4 AIM Inputs

The following list outlines the Masstran inputs along with their default value available through the AIM interface.

- **Tess_Params = [0.025, 0.001, 15.0]**
  Body tessellation parameters used when creating a boundary element model. Tess_Params[0] and Tess↩_Params[1] get scaled by the bounding box of the body. (From the EGADS manual) A set of 3 parameters that drive the EDGE discretization and the FACE triangulation. The first is the maximum length of an EDGE segment or triangle side (in physical space). A zero is flag that allows for any length. The second is a curvature-based value that looks locally at the deviation between the centroid of the discrete object and the underlying geometry. Any deviation larger than the input value will cause the tessellation to be enhanced in those regions. The third is the maximum interior dihedral angle (in degrees) between triangle facets (or Edge segment tangents for a WIREBODY tessellation), note that a zero ignores this phase

- **Edge_Point_Min = 2**
  Minimum number of points on an edge including end points to use when creating a surface mesh (min 2).

- **Edge_Point_Max = 50**
  Maximum number of points on an edge including end points to use when creating a surface mesh (min 2).

- **Quad_Mesh = False**
  Create a quadratic mesh on four edge faces when creating the boundary element model.

- **Property = NULL**
  Property tuple used to input property information for the model, see FEA Property for additional details.

- **Material = NULL**
  Material tuple used to input material information for the model, see FEA Material for additional details.

- **Surface_Mesh = NULL**
  A Surface_Mesh link.

- **Design_Variable = NULL**
  The design variable tuple is used to input design variable information for the model optimization, see FEA Design Variables for additional details.

- **Design_Variable_Relation = NULL**
  The design variable relation tuple is used to input design variable relation information for the model optimization, see FEA DesignVariableRelation for additional details.

0.5 AIM Outputs

The following list outlines the Masstran outputs available through the AIM interface.

- **Area** = Total area of the mesh.
- **Mass** = Total mass of the model.
- **Centroid** = Centroid of the model.
- **CG** = Center of gravity of the model.
- **Ixx** = Moment of inertia
- **Iyy** = Moment of inertia
- **Izz** = Moment of inertia
- **Ixy** = Moment of inertia

- **Izy** = Moment of inertia
- **Iyz** = Moment of inertia
- **I_Vector** = Moment of inertia vector

$$\vec{I} = \begin{bmatrix} I_{xx} & I_{yy} & I_{zz} & I_{xy} & I_{xz} & I_{yz} \end{bmatrix}$$

- **I_Lower** = Moment of inertia lower triangular tensor

$$\vec{I}_{lower} = \begin{bmatrix} I_{xx} & -I_{xy} & I_{yy} & -I_{xz} & -I_{yz} & I_{zz} \end{bmatrix},$$

- **I_Upper** = Moment of inertia upper triangular tensor

$$\vec{I}_{upper} = \begin{bmatrix} I_{xx} & -I_{xy} & -I_{xz} & I_{yy} & -I_{yz} & I_{zz} \end{bmatrix},$$

- **I_Tensor** = Moment of inertia tensor

$$\bar{\bar{I}} = \begin{bmatrix} I_{xx} & -I_{xy} & -I_{xz} \\ -I_{xy} & I_{yy} & -I_{yz} \\ -I_{xz} & -I_{yz} & I_{zz} \end{bmatrix}$$

0.6 FEA Material

Structure for the material tuple = ("Material Name", "Value"). "Material Name" defines the reference name for the material being specified. The "Value" can either be a JSON String dictionary (see Section JSON String Dictionary) or a single string keyword (see Section Single Value String).

0.6.1 JSON String Dictionary

If "Value" is JSON string dictionary (e.g. "Value" = {"density": 7850, "youngModulus": 120000.0, "poissonRatio": 0.5, "materialType": "isotropic"}) the following keywords (= default values) may be used:

- **materialType = "Isotropic"**
  Material property type. Options: Isotropic.

- **density = 0.0**
  Density of the material.

0.6.2 Single Value String

If "Value" is a string, the string value may correspond to an entry in a predefined material lookup table. NOT YET IMPLEMENTED!!!!

0.7 FEA Property

Structure for the property tuple = ("Property Name", "Value"). "Property Name" defines the reference `capsGroup` for the property being specified. The "Value" can either be a JSON String dictionary (see Section JSON String Dictionary) or a single string keyword (see Section Single Value String).

0.7.1 JSON String Dictionary

If "Value" is JSON string dictionary the following keywords (= default values) may be used:

- **propertyType = No Default value**
  Type of property to apply to a given capsGroup `Name`. Options: ConcentratedMass, Shell

- **membraneThickness = 0.0**
  Membrane thickness.

- **massPerArea = 0.0**
  Mass per unit area.

- **massOffset = [0.0, 0.0, 0.0]**
  Offset distance from the grid point to the center of gravity for a concentrated mass.

- **massInertia = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0]**
  Mass moment of inertia measured at the mass center of gravity.

0.7.2 Single Value String

If "Value" is a string, the string value may correspond to an entry in a predefined property lookup table. NOT YET IMPLEMENTED!!!!

0.8 FEA Constraint

Structure for the constraint tuple = ("Constraint Name", "Value"). "Constraint Name" defines the reference name for the constraint being specified. The "Value" can either be a JSON String dictionary (see Section JSON String Dictionary) or a single string keyword (see Section Single Value String).

0.8.1 JSON String Dictionary

If "Value" is JSON string dictionary the following keywords (= default values) may be used:

0.8.2 Single Value String

If "Value" is a string, the string value may correspond to an entry in a predefined constraint lookup table. NOT YET IMPLEMENTED!!!!

0.9 FEA Support

Structure for the support tuple = ("Support Name", "Value"). "Support Name" defines the reference name for the support being specified. The "Value" can either be a JSON String dictionary (see Section JSON String Dictionary) or a single string keyword (see Section Single Value String).

0.9.1 JSON String Dictionary

If "Value" is JSON string dictionary the following keywords (= default values) may be used:

0.9.2 Single Value String

If "Value" is a string, the string value may correspond to an entry in a predefined support lookup table. NOT YET IMPLEMENTED!!!!

0.10 FEA Connection

Structure for the connection tuple = ("Connection Name", "Value"). "Connection Name" defines the reference name to the capsConnect being specified and denotes the "source" node for the connection. The "Value" can either be a JSON String dictionary (see Section JSON String Dictionary) or a single string keyword (see Section Single Value String).

0.10.1 JSON String Dictionary

If "Value" is JSON string dictionary the following keywords (= default values) may be used:

0.10.2 Single Value String

If "Value" is a string, the string value may correspond to an entry in a predefined connection lookup table. NOT YET IMPLEMENTED!!!!

0.11 FEA Load

Structure for the load tuple = ("Load Name", "Value"). "Load Name" defines the reference name for the load being specified. The "Value" can either be a JSON String dictionary (see Section JSON String Dictionary) or a single string keyword (see Section Single Value String).

0.11.1 JSON String Dictionary

If "Value" is JSON string dictionary the following keywords (= default values) may be used:

0.11.2 Single Value String

If "Value" is a string, the string value may correspond to an entry in a predefined load lookup table. NOT YET IMPLEMENTED!!!!

0.12 FEA Analysis

Structure for the analysis tuple = ('Analysis Name', 'Value'). 'Analysis Name' defines the reference name for the analysis being specified. The "Value" can either be a JSON String dictionary (see Section JSON String Dictionary) or a single string keyword (see Section Single Value String).

0.12.1 JSON String Dictionary

If "Value" is JSON string dictionary the following keywords (= default values) may be used:

0.12.2 Single Value String

If "Value" is a string, the string value may correspond to an entry in a predefined analysis lookup table. NOT YET IMPLEMENTED!!!!

0.13 FEA Design Variables

Structure for the design variable tuple = ("DesignVariable Name", "Value"). "DesignVariable Name" defines the reference name for the design variable being specified. This string will be used in the FEA input directly. The "Value" must be a JSON String dictionary (see Section JSON String Dictionary).

0.13.1 JSON String Dictionary

If "Value" is JSON string dictionary (eg. "Value" = {"initialValue": 5.0, "upperBound": 10.0}) the following keywords (= default values) may be used:

- **initialValue = 0.0**
  Initial value for the design variable.

0.14 FEA DesignVariableRelation

Structure for the design variable tuple = ("DesignVariableRelation Name", "Value"). "DesignVariableRelation Name" defines the reference name for the design variable being specified. This string will be used in the FEA input directly. The "Value" must be a JSON String dictionary (see Section JSON String Dictionary).

0.14.1 JSON String Dictionary

If "Value" is JSON string dictionary (eg. "Value" = {"componentType": "Property", "componentName": "plate", "fieldName": "TM", "variableName": "MyDesVar"}) the following keywords (= default values) may be used:

- **componentType = "Property"**
  The type of component for this design variable relation. Options: "Material", "Property", "Element".

- **componentName = "(no default)"**
  Single or list of FEA Property(ies), or FEA Material name(s) linked to the design variable relation (e.g. "↩ Name1" or ["Name1","Name2",...].
  - For `componentType` Property a FEA Property name (or names) is given.
  - For `componentType` Material a FEA Material name (or names) is given.

- **variableName = "(no default)"**
  Single or list of names of design variables linked to this relation

- **fieldName = "(no default)"**
  Fieldname of variable relation (e.g. "E" for Young's Modulus). Design Variable Relations can be defined as three types based on the `variableType` value. These are Material, Property, or Element. This means that an aspect of a material, property, or element input can change in the optimization problem. This input specifies what aspect of the Material, Property, or Element is changing.
  1. **Material Types** Selected based on the material type (see FEA Material, materialType) referenced in the `componentName` above.
     - `materialType = "Isotropic"`
       * "density"
  2. **Property Types** (see FEA Property)
     - `propertyType = "ConcentratedMass"`
       * "mass",
       * "massOffset1", "massOffset2", "massOffset2"
       * "Ixx", "Iyy", "Izz", "Ixy", "Ixz", "Iyz"
     - `propertyType = "Shell"`
       * "membraneThickness", "massPerArea"

- **constantCoeff = 0.0**
  Constant term of relation.

- **linearCoeff = 1.0**
  Single or list of coefficients of linear relation. Must be same length as `variableName`.

0.15 FEA Design Constraints

Structure for the design constraint tuple = ('DesignConstraint Name', 'Value'). 'DesignConstraint Name' defines the reference name for the design constraint being specified. The "Value" must be a JSON String dictionary (see Section JSON String Dictionary).

0.15.1 JSON String Dictionary

If "Value" is JSON string dictionary the following keywords (= default values) may be used:

0.16 FEA Optimization Control

Structure for the optimization control dictionary = 'Value'. The "Value" must be a JSON String dictionary (see Section JSON String Dictionary).

0.17 FEA Design Equations

Structure for the design equation tuple = ("DesignEquation Name", ["Value1", ... , "ValueN"]). "DesignEquation Name" defines the reference name for the design equation being specified. This string will be used in the FEA input directly. The values "Value1", ... , "ValueN" are a list of strings containing the equation defintions. (see Section List of equation strings).

0.17.1 List of equation strings

Each design equation tuple value is a list of strings containing the equation definitions

0.18 FEA Table Constants

Structure for the table constant tuple = ("TableConstant Name", "Value"). "TableConstant Name" defines the reference name for the table constant being specified. This string will be used in the FEA input directly. The "Value" is the value of the table constant.

0.19 FEA Design Responses

Structure for the design response tuple = ("DesignResponse Name", "Value"). "DesignResponse Name" defines the reference name for the design response being specified. This string will be used in the FEA input directly. The "Value" must be a JSON String dictionary (see Section JSON String Dictionary).

0.19.1 JSON String Dictionary

If "Value" is JSON string dictionary the following keywords (= default values) may be used:

0.20 FEA Design Equation Responses

Structure for the design equation response tuple = ("DesignEquationResponse Name", "Value"). "DesignEquation←↩ Response Name" defines the reference name for the design equation response being specified. This string will be used in the FEA input directly. The "Value" must be a JSON String dictionary (see Section JSON String Dictionary).

0.20.1 JSON String Dictionary

If "Value" is JSON string dictionary the following keywords (= default values) may be used:

0.21 FEA Design Optimization Parameters

Structure for the design optimization parameter tuple = ("DesignOptParam Name", "Value"). "DesignOptParam Name" defines the reference name for the design optimization parameter being specified. This string will be used in the FEA input directly. The "Value" is the value of the design optimization parameter.

0.22 FEA Aerodynamic References

The aerodynamic reference input must be a JSON String dictionary (see Section JSON String Dictionary).

0.22.1 JSON String Dictionary

The following keywords (= default values) may be used:

0.23 Masstran AIM Basic Example

This is a walkthrough for using Masstran AIM to analyze a three-dimensional wing with internal ribs and spars.

0.23.1 Prerequisites

It is presumed that ESP and CAPS have been already installed, as well as Masstran.

0.23.1.1 Script files

Two scripts are used for this illustration:

1. feaWingBEM.csm: Creates geometry, as described in the next section (Creating Geometry using ESP).
2. masstran_PyTest.py: pyCAPS script for performing analysis, as described in Performing analysis using pyCAPS
.

0.23.2 Creating Geometry using ESP

The CSM script generates Bodies which are designed to be used by specific AIMs. The AIMs that the Body is designed for is communicated to the CAPS framework via the "capsAIM" string attribute. This is a semicolon-separated string with the list of AIM names. Thus, the CSM author can give a clear indication to which AIMs should use the Body. In this example, the list contains the structural finite element analysis tools that can analyze the body:

```
attribute capsAIM $nastranAIM;astrosAIM;mystranAIM;masstranAIM;egadsTessAIM
```

A typical geometry model can be created and interactively modified using design parameters. These design parameters are either design- or geometry- based. In this example, a wing configuration is created using following design parameters.

```
# Design Parameters for OML
despmtr   thick     0.12      frac of local chord
despmtr   camber    0.04      frac of loacl chord
despmtr   area      10.0
despmtr   aspect    6.00
despmtr   taper     0.60
despmtr   sweep     20.0      deg (of c/4)
despmtr   washout   5.00      deg (down at tip)
despmtr   dihedral  4.00      deg
# Design Parameters for BEM
cfgpmtr   nrib      11        number of ribs
despmtr   spar1     0.20      frac of local chord
despmtr   spar2     0.75      frac of local chord
```

After our design parameters are defined they are used to setup other local variables (analytically) for the outer model line (OML).

```
# OML
set       span      sqrt(aspect*area)
set       croot     2*area/span/(1+taper)
set       ctip      croot*taper
set       dxtip     (croot-ctip)/4+span/2*tand(sweep)
set       dytip     span/2*tand(dihedral)
```

In a similar manner, local variables are defined for the ribs and spars.

```
# wing ribs
set       Nrib    nint(nrib)
# wing spars
set       eps             0.01*span
```

Once all design and local variables are defined, a full span, solid model is created by "ruling" together NACA series airfoils (following a series of scales, rotations, and translations).

```
mark
   # Right tip
   udprim    naca      Thickness thick     Camber    camber
   scale     ctip
   rotatez   washout   ctip/4    0
   translate dxtip     dytip     -span/2
   # root
   udprim    naca      Thickness thick     Camber    camber
   scale     croot
   # left tip
   udprim    naca      Thickness thick     Camber    camber
   scale     ctip
   rotatez   washout   ctip/4    0
   translate dxtip     dytip     +span/2
rule
   attribute OML 1
```

Once complete, the wing is stored for later use under the name OML.

```
store     OML
```

Next, the inner layout of the ribs and spars are created using the waffle udprim.

```
udprim    waffle    Depth     +6*thick*croot      Filename «
   patbeg    i Nrib
      point A   at (span/2)*(2*i-Nrib-1)/Nrib   -0.01*croot
      point B   at (span/2)*(2*i-Nrib-1)/Nrib   max(croot,dxtip+ctip)
      line  AB  A  B   tagComponent=rib tagIndex=!val2str(i,0)
   patend
   point A   at -span/2-eps   spar1*ctip+dxtip
   point B   at 0             spar1*croot
   line  AB  A  B   tagComponent=spar tagIndex=1 tagPosition=left
   point A   at span/2+eps    spar1*ctip+dxtip
   point B   at 0             spar1*croot
   line  AB  A  B    tagComponent=spar tagIndex=1 tagPosition=right
   point A   at -span/2-eps   spar2*ctip+dxtip
```

```
   point B   at 0             spar2*croot
   line  AB  A  B   tagComponent=spar tagIndex=2 tagPosition=left
   point A   at span/2+eps    spar2*ctip+dxtip
   point B   at 0             spar2*croot
   line  AB  A  B   tagComponent=spar tagIndex=2 tagPosition=right
»
```

An attribute is then placed on ribs and spars so that the geometry components may be reference by the Masstran AIM.

```
attribute capsGroup $Ribs_and_Spars
```

Following a series of rotations and translations the ribs and spars are stored for later use.

```
translate 0         0         -3*thick*croot
rotatey   90        0         0
rotatez  -90        0         0
store     layoutRibSpar
```

Next, the layout of the ribs and spars are intersected the outer mold line of wing, which results in only keeping the part of layout that is inside the OML.

```
restore   layoutRibSpar
restore   OML
intersect
```

Finally, select faces (airfoil sections at the root) are tagged, so that a constraint may be applied later.

```
udprim editAttr filename «
   edge adj2face tagComponent=spar tagPosition=right
   and  adj2face tagComponent=spar tagPosition=left
   set  capsConstraint=Rib_Constraint
   node adj2face tagComponent=spar tagPosition=right
   and  adj2face tagComponent=spar tagPosition=left
   set  capsConstraint=Rib_Constraint
»
ifthen nint(mod(Nrib,2)) ne 0
   set       midRib Nrib/2
   select    face   $tagComponent $rib $tagIndex val2str(midRib,0)
   attribute tagPosition $root

   udprim editAttr filename «
      face   has tagComponent=rib tagPosition=root
      set    capsConstraint=Rib_Constraint

      edge   adj2face tagComponent=rib tagPosition=root
      set    capsConstraint=Rib_Constraint

      node   adj2face tagComponent=rib tagPosition=root
      set    capsConstraint=Rib_Constraint
»
endif
```

The above *.csm file results in the follow geometry model:

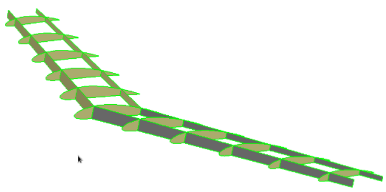

Figure 1 Wing built up element model

0.23.3 Performing analysis using pyCAPS

The first step in the pyCAPS script is to import the required modules. For this example the following modules are used,

```
import pyCAPS
import os
import argparse
```

Once the required modules have been loaded, a pyCAPS.Problem can be instantiated with the desired geometry file.

```
geometryScript = os.path.join("..","csmData","feaWingBEM.csm")
```

```
myProblem = pyCAPS.Problem(problemName=workDir,
                           capsFile=geometryScript,
                           outLevel=args.outLevel)
```

After the geometry is loaded, a structural mesh is generated using the egadsTessAIM.

```
# Load egadsTess aim
myProblem.analysis.create( aim = "egadsTessAIM",
                           name = "tess" )
# All triangles in the grid
myProblem.analysis["tess"].input.Mesh_Elements = "Quad"
# Set global tessellation parameters
myProblem.analysis["tess"].input.Tess_Params = [.05,.5,15]
# The surfaces mesh is generated automatically just-in-time
```

Next, the Masstran AIM is instantiated.

```
masstranAIM = myProblem.analysis.create(aim = "masstranAIM",
                                        name = "masstran")
```

Once loaded analysis parameters specific to Masstran need to be set (see AIM Inputs). Here, the mesh from the surface egadsTessAIM is linked to the masstranAIM.

```
masstranAIM.input["Surface_Mesh"].link(myProblem.analysis["tess"].output["Surface_Mesh"])
```

Note the AIM instances are referenced in two different manners:

1. Using the returned object from Problem.analysis.create call.
2. Using the "name" key in the Problem.analysis Sequence. While syntactically different, these two forms are essentially identical.

Along the same lines of setting the input values above the "Material" (see FEA Material) and "Property" (see FEA Property) dictionaries are used to set more complex information. The user is encouraged to read the additional documentation on these inputs for further explanations.

```
# Set materials
unobtainium  = {"youngModulus" : 2.2E11 ,
                "poissonRatio" : .33,
                "density"      : 7850}
madeupium    = {"materialType" : "isotropic",
                "youngModulus" : 1.2E9 ,
                "poissonRatio" : .5,
                "density"      : 7850}
masstranAIM.input.Material = {"Unobtainium": unobtainium,
                              "Madeupium"  : madeupium}
# Set property
shell  = {"propertyType"       : "Shell",
          "membraneThickness" : 0.2,
          "bendingInertiaRatio" : 1.0, # Default
          "shearMembraneRatio"  : 5.0/6.0} # Default }
masstranAIM.input.Property = {"Ribs_and_Spars": shell}
```

The MasstrainAIM will execute automatically and compute all mass properties in memory when an output is reqested below.

Finally, available AIM outputs (see AIM Outputs) may be retrieved, for example:

```
# Get mass properties
print ("\nGetting results mass properties.....\n")
Area     = masstranAIM.output.Area
Mass     = masstranAIM.output.Mass
Centroid = masstranAIM.output.Centroid
CG       = masstranAIM.output.CG
Ixx      = masstranAIM.output.Ixx
Iyy      = masstranAIM.output.Iyy
Izz      = masstranAIM.output.Izz
Ixy      = masstranAIM.output.Ixy
Ixz      = masstranAIM.output.Ixz
Iyz      = masstranAIM.output.Iyz
I        = masstranAIM.output.I_Vector
II       = masstranAIM.output.I_Tensor
print("Area     ", Area)
print("Mass     ", Mass)
print("Centroid ", Centroid)
print("CG       ", CG)
print("Ixx      ", Ixx)
print("Iyy      ", Iyy)
print("Izz      ", Izz)
print("Ixy      ", Ixy)
```

```
print("Ixz      ", Ixz)
print("Iyz      ", Iyz)
print("I        ", I)
print("II       ", II)
```

results in,

```
Area      3.28946557
Mass      5164.46094491
Centroid  [1.2409841844368583, 0.16359702451265337, 4.0874212239589455e-09]
CG        [1.2409841844368585, 0.16359702451265348, 4.087420991763218e-09]
Ixx       21325.300951
Iyy       22558.0731769
Izz       1292.98036179
Ixy       153.720903861
Ixz       2.06373216532e-06
Iyz       2.36311990987e-06
I         [21325.300951015903, 22558.07317691867, 1292.9803617927173, 153.72090386142395,
       2.063732165317917e-06, 2.363119909871653e-06]
II        [[21325.300951015903, -153.72090386142395, -2.063732165317917e-06], [-153.72090386142395,
       22558.07317691867, -2.363119909871653e-06], [-2.063732165317917e-06, -2.363119909871653e-06,
       1292.9803617927173]]
```

0.23.4 Executing pyCAPS script

Issuing the following command executes the script:

```
python masstran_PyTest.py
```