

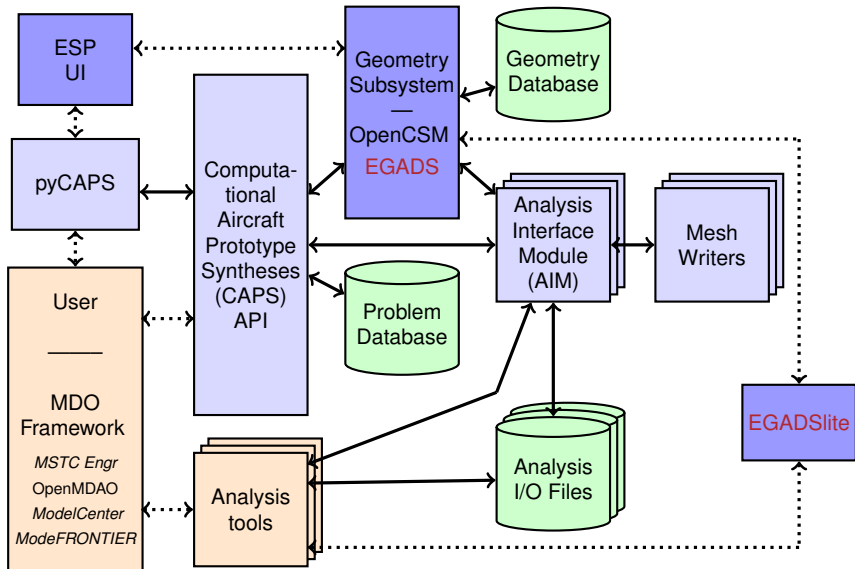


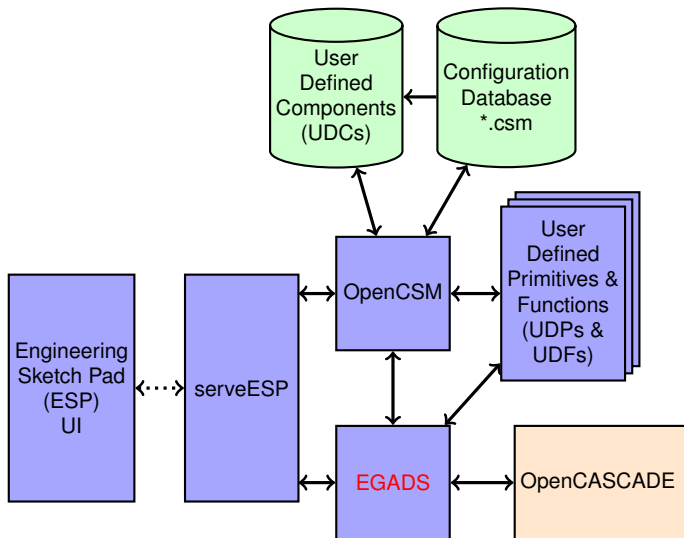
The Use of Geometry from within the Engineering Sketch Pad – Rev 1.29 The EGADS API 1/2

Bob Haimes

bob@geocentrictech.com or haimes@mit.edu

Geocentric Technologies LLC





The Engineering Geometry Aircraft Design System (EGADS) is an open-source geometry interface to OpenCASCADE

- reduces OpenCASCADE's 17,000 methods to about 100 calls
- supports programming in C, C++, FORTRAN, Python and Julia
- allow *bottom-up* construction via geometric and topological primitives
- allows *top-down* construction via solid creation and Boolean operations
- provides persistent user-defined attributes on topological entities
- adjustable tessellator (vs a surface mesher) with support for finite-differencing in the calculation of parametric sensitivities

EGADSlite – for HPC Environments

- No construction supported
- Same API and Object model as EGADS
 - Can use EGADS to prototype/build EGADSlite code
- Suitable for an MPI setup:
 - Data export from EGADS via a *stream*
 - Data import to EGADSlite from the *stream*
 - *Stream* setup to Broadcast (or write to disk)
- ANSI C – No OpenCASCADE
- Tiny memory footprint
- Thread safe and scalable
 - EGADS' OpenCASCADE evaluation functions replaced with those written for EGADSlite

See [\\$ESP_ROOT/externApps/Pagoda/EGADSserver](#) for an MPI example

System Support

- MacOS (Intel or Mx) with **clang**, **ifort/ifx** and/or **gfortran**
- LINUX with **gcc**, **ifort/ifx** and/or **gfortran**
- Windows with Microsoft Visual Studio C++ and **ifort/ifx**
- No globals (but not entirely thread-safe due to OpenCASCADE)
- Various levels of output (0-none, through 3-debug)
- Written in C and C++
- pyEGADS only requires a current version of Python

EGADS Objects (**egos**)

- Pointer to a C structure – allows for an *Object-based* API
- Treated as “blind” pointers (i.e., not meant to be dereferenced)
- **egos** are INTEGER*8 variables in FORTRAN

- Context – Holds the *globals*
- Transform
- Tessellation
- Nil (allocated but not assigned) – internal
- Empty – internal
- Reference – internal
- Geometry
 - pcurve, curve, surface
- Topology
 - Node, Edge, Loop, Face, Shell, Body, Model

See [\\$ESP_ROOT/doc/EGADS/egads.pdf](#) for a detailed description of all of the functions.

See [\\$ESP_ROOT/include/egadsTypes.h](#) for a list of **defines** and structures.

See [\\$ESP_ROOT/include/egadsErrors.h](#) for a list of the return code **defines**.

C structure definition - an ego

```
typedef struct egObject {  
    int      magicnumber;      /* must be properly set to validate  
                               the object */  
  
    short    oclass;           /* object class */  
    short    mtype;           /* object member type */  
    void     *attrs;           /* attributes or reference */  
    void     *blind;           /* blind pointer to OpenCASCADE or  
                               EGADS data */  
  
    struct egObject *topObj;    /* top of the hierarchy or  
                               context (if top) */  
  
    struct egObject *ref;       /* threaded list of references */  
    struct egObject *prev;      /* back pointer */  
    struct egObject *next;      /* forward pointer */  
} egObject;  
  
#define ego egObject*;
```

Context Object

- Start of dual threaded-list of active **egos**
- Pool of deleted objects

Deleting Objects

- Use the function `EG_deleteObject` to delete Objects
- The Object must be reference *free* – i.e. not used by another
 - Delete in the opposite order of creation
 - If in a Body, delete the Body instead (unless the Body is in a Model)
- `EG_deleteObject` on a Context does not delete the Context
 - Deletes all Objects in the Context that are not in a Body
 - Use `EG_close` to delete all objects in a Context (and the Context)

Another Rule

- A Body can only be in one Model
 - Copy the Body of interest, then include the copy in the new Model

surface

- 3D surfaces in the space of 2 parameters: $[u, v]$
- **Types:** Plane, Spherical, Cylindrical, Revolution, Toriodal, Trimmed, Bezier, BSpline, Offset, Conical, Extrusion
- All types abstracted to $[x, y, z] = f(u, v)$

pcurve – Parameter Space Curves

- 2D curves in the Parametric space $[u, v]$ of a surface
- **Types:** Line, Circle, Ellipse, Parabola, Hyperbola, Trimmed, Bezier, BSpline, Offset
- All types abstracted to $[u, v] = h(t)$

curve

- 3D curve in the space of 1 running parameter: t
- **Types:** Line, Circle, Ellipse, Parabola, Hyperbola, Trimmed, Bezier, BSpline, Offset
- All types abstracted to $[x, y, z] = g(t)$

- All EGADS C/C++ Functions begin with “EG_”
- There is an attempt to have a descriptive function name
- Inputs are usually at the beginning of the argument list
- Outputs are usually at the end
- Return Values (*icode*):
 - (Almost) all EGADS function have an *icode* return value
 - A value of 0 (EGADS_SUCCESS) indicates success
 - A negative value indicates an error
 - see `$ESP_ROOT/include/egadsErrors.h` for a list of the return code *defines*.
 - Some functions have a positive return code to indicate partial success or provide other information to the caller

Create a Geometry Object

```
icode = EG_makeGeometry(ego context, int oclass, int mtype, ego rGeom,  
                        const int *ints, const double *reals,  
                        ego *nGeom);
```

context the Context Object

oclass the Object Class: PCURVE, CURVE or SURFACE

mtype the Member Type (depends on **oclass**)

rGeom the reference Geometry Object (if none use **NULL**)

ints the integer information (if none use **NULL**)

reals the real data used to construct the geometry

nGeom the returned pointer to the new Geometry Object

icode the integer return code

Notes:

- 1 **ints** is required for either **mtype** = BEZIER or BSPLINE
- 2 See pages 16-29 of [\\$ESP_ROOT/doc/EGADS/egads.pdf](#) for **oclass/mtype** data requirements

Evaluating the Object

```
icode = EG_evaluate(ego object, double *params, double *result);
```

object the input Object

params NODE – ignored (can be NULL)
 PCURVE, CURVE, EDGE – the t value
 SURFACE, FACE – u then v

result the filled returned position, 1st and 2nd derivatives:

length $\Rightarrow$	Node – 3	PCurve – 6	Edge Curve – 9	Face Surface – 18
Position	$[x, y, z]$	$[u, v]$	$[x, y, z]$	$[x, y, z]$
1 st	–	$[du, dv]$	$[dx, dy, dz]$	$[dx_u, dy_u, dz_u]$ $[dx_v, dy_v, dz_v]$
2 nd	–	$[du^2, dv^2]$	$[dx^2, dy^2, dz^2]$	$[dx_u^2, dy_u^2, dz_u^2]$ $[dx_{uv}, dy_{uv}, dz_{uv}]$ $[dx_v^2, dy_v^2, dz_v^2]$

icode the integer return code

Note: You cannot evaluate a DEGENERATE Edge.

Inverse evaluation on the Object

```
icode = EG_invEvaluate(ego object, double *pos, double *params,  
                      double *result);
```

object the input Object

pos is $[u, v]$ for a PCURVE and $[x, y, z]$ for all others

params the returned parameter(s) found for the nearest position on the Object:
for PCURVE, CURVE or EDGE the one value is t
for SURFACE or FACE the 2 values are u then v

result the closest position found is returned:
 $[u, v]$ for a PCURVE (len = 2)
 $[x, y, z]$ for all others (len = 3)

icode the integer return code

Note: When using this with a Face the timing is significantly slower than making the call with the Face's reference surface (due to the clipping). If you don't need this limiting call `EG_invEvaluate` with the underlying Surface Object.

Boundary Representation – BRep

<i>Top</i> <i>Down</i> ↓ ↑ <i>Bottom</i> <i>Up</i>	Topology	Geometric Entity	Function
	Model		
	Body	Solid, Sheet, Face, Wire	
	Shell		
	Face	surface	$(x, y, z) = \mathbf{f}(u, v)$
	Loop	pcurve (non-planar)	
	Edge	curve	$(x, y, z) = \mathbf{g}(t)$
	Node	point	

- Nodes that bound Edges may not be exactly on the underlying curves
- Edges in the Loops that trim the Face may not exactly sit on the surface, hence the use of pcurves

Node

- Contains $[x, y, z]$
- Types: none

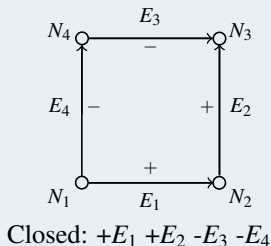
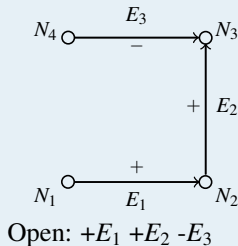
Edge

- Has a 3D curve (if not Degenerate)
- Has a t range (t_{min} to t_{max} , where $t_{min} < t_{max}$)
Note: The positive orientation is going from t_{min} to t_{max}
- Has a Node for t_{min} and for t_{max} – can be the same Node
- Types:
 - OneNode – periodic
 - TwoNode – normal
 - Degenerate – single Node, t range used for the associated pcurve



Loop – without a reference surface

- 1 Free standing connected Edges that can be used in a non-manifold setting (for example in WireBodies)
- 2 A list of connected Edges associated with a Plane (which does not require pcurves)
 - An ordered collection of Edge objects with associated senses
 - Edges must not be Degenerate
 - Types:



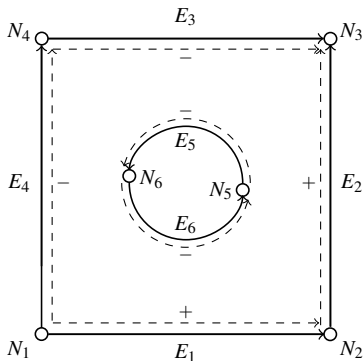
Loop – with a reference surface

- ① Collections of Edges followed by a corresponding collection of pcurves that define the $[u, v]$ trimming on the surface
- An ordered collection of Edge objects with associated senses
- Degenerate Edges are required when the $[u, v]$ mapping collapses like at the apex of a cone (note that the pcurve is needed to be fully defined using the Edge's t range)
- Trims the surface by maintaining material to the left of the running Loop
- An Edge may be found in a Loop twice (with opposite senses) and with different pcurves.
- Types: Open or Closed (comes back on itself)

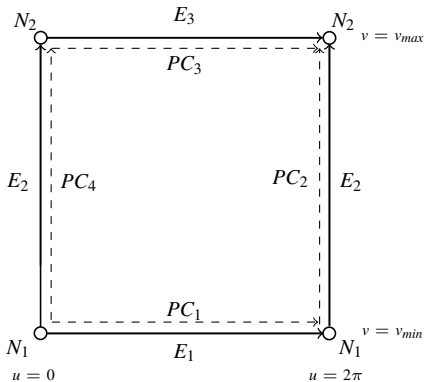
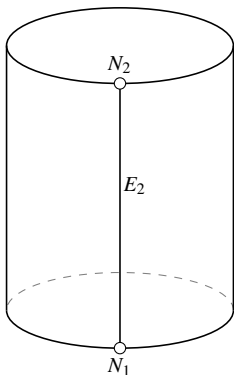
Face

- A surface bounded by one or more Loops with associated senses
- Only one outer Loop (sense = 1) and any number of inner Loops (sense = -1). Note that under very rare conditions a Loop may be found in more than 1 Face – in this case the one marked with sense = +/- 2 must be used in a reverse manner.
- All Loops must be Closed
- Loop(s) must not contain reference geometry for Planar surfaces
- If the surface is not a Plane then the Loop's reference Object must match that of the Face
- Type is the orientation of the Face based on surface's $\vec{U} \times \vec{V}$:
 - SFORWARD or SREVERSE when the orientations are opposed

Note that this is coupled with the Loop's orientation (i.e. an outer Loop traverses the Face in a right-handed manner defining the outward direction)



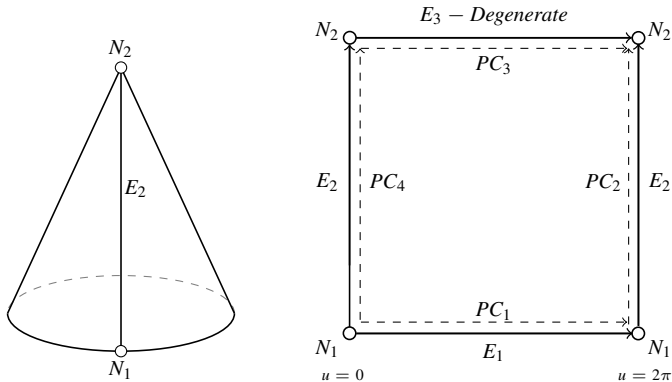
- Outer Loop – right handed/counterclockwise: $+E_1 +E_2 -E_3 -E_4$
- Inner Loop – left handed/clockwise: $-E_5 -E_6$



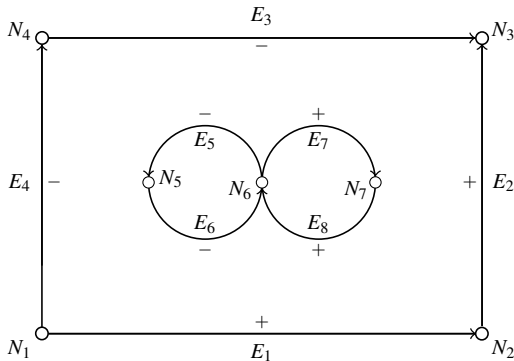
Unrolled periodic cylinder Face

Single Outer Loop – right handed/counterclockwise:

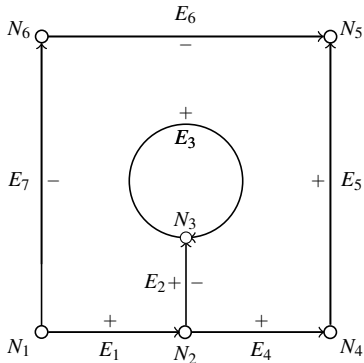
$$+E_1 +E_2 -E_3 -E_2$$



Unrolled Cone



- Outer Loop – right handed/counterclockwise: $+E_1 +E_2 -E_3 -E_4$
- Inner Loop #1 – left handed/clockwise: $-E_5 -E_6$
- Inner Loop #2 – left handed/clockwise: $+E_7 +E_8$



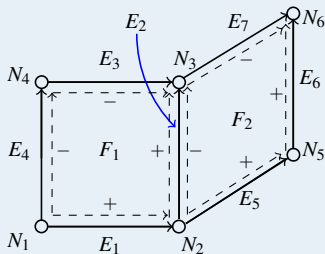
Single Outer Loop – right handed/counterclockwise:

$$+E_1 +E_2 +E_3 -E_2 +E_4 +E_5 -E_6 -E_7$$

Note: PCurve the same for both sides of E_2

Shell

- A collection of one or more connected Faces, that if Closed segregates regions of 3-Space
- All Faces must be properly oriented
- Non-manifold Shells can have more than 2 Faces sharing an Edge
- Types: Open (including non-manifold) or Closed

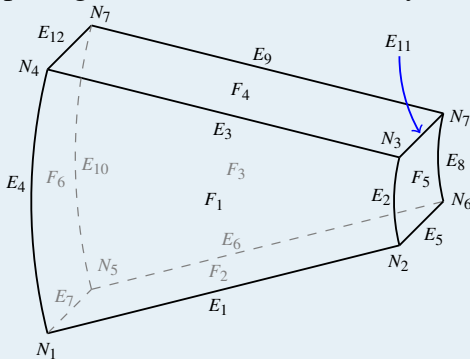


Face #1 Loop: $+E_1 +E_2 -E_3 -E_4$

Face #2 Loop: $+E_5 +E_6 -E_7 -E_2$

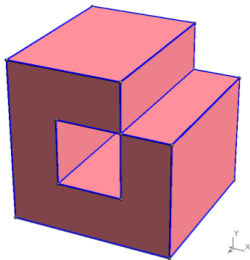
SolidBody

- Manifold collection of one or more Closed Shells
- One outer Shell (sense = 1); any number of inner (sense = -1)
- Edges (except Degenerate) are found exactly twice (sense = ± 1)

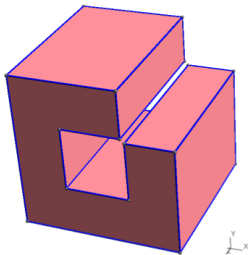


Simple SolidBody: 8 Nodes, 12 Edges, 6 Loops and 6 Faces

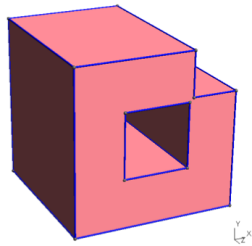
Manifold vs. Nonmanifold



nonmanifold



manifold



manifold

figure stolen from "An introduction to Geometrical Modelling and Mesh Generation: The Gmsh Companion" by Christophe Geuzaine, Emilie Marchandise & Jean-François Remacle – used without permission!

Can the geometry be manufactured?

Body – including SolidBody

- Container used to aggregate Topology
- Connected to support non-manifold collections at the Model level
- *Owns* all the Objects contained within it
- Types:
 - A WireBody contains a single Loop
 - A FaceBody contains a single Face – IGES import
 - A SheetBody contains a single Shell which can be either non-manifold or manifold (though usually a manifold Body of this type is promoted to a SolidBody)

Model

- A collection of Bodies – becomes the *Owner* of contained Objects
- Returned by SBO & Sew Functions
- Read and Written by EGADS

Create a Topology Object

```
icode = EG_makeTopology(ego context, ego geom, int oclass, int mtype,  
                        double *reals, int nchild, ego *children,  
                        int *senses, ego *topo);
```

context the Context Object

geom the reference Geometry Object (if none use **NULL**)

oclass the Object Class: NODE, EDGE, LOOP, FACE, SHELL, BODY or MODEL

mtype the Member Type (depends on **oclass**)

reals the real data: may be **NULL** except for NODE that contains the $[x, y, z]$ location and EDGE where the t_{min} and t_{max} (the parametric bounds) are specified

nchild number of children (lesser) Topological Objects

children vector of children objects (**nchild** in length)
if a LOOP with a reference SURFACE, then $2 * \text{nchild}$ in length (PCurves follow)

senses a vector of children integer senses: SFORWARD/SREVERSE for LOOP, and SOUTER/SINNER for FACE **nchild** > 1 (may be **NULL** for FACE **nchild** = 1)

topo the returned pointer to the new Topology Object

icode the integer return code

Query a Topology Object

```
icode = EG_getTopology(ego topo, ego *geom, int *oclass, int *mtype,  
                      double *reals, int *nchild, ego **children,  
                      int **senses);
```

- topo** the Topology or *Effective Topology* Object to query
- geom** the returned reference Geometry Object (can be **NULL**)
- oclass** the returned Topology Object Class
- mtype** the returned Member Type (depends on **oclass**)
- reals** the real data (at most 4 **doubles** are filled): **NODE** – contains the $[x, y, z]$ location, **EDGE** where the t_{min} and t_{max} (the parametric bounds) are returned and **FACE** where the $[u, v]$ box is filled $\rightarrow$ the limits first for u then for v (4 in length)
- nchild** the returned number of children (lesser) Topological Objects
- children** the returned pointer to a vector of children objects (**nchild** in length)
 - if a **LOOP** with a reference **SURFACE**, then $2 * \text{nchild}$ in length (**PCurves** follow)
 - if a **MODEL** – **nchild** is the number of Body Objects, **mtype** the total **ego** count
- senses** a vector of senses for the children (**LOOPS**) or inner/outer for (**FACES** & **SHELLS**)
- icode** the integer return code

Get revision

```
EG_revision(int *imajor, int *iminor, const char **OCCrev);
```

imajor the returned major revision

iminor the returned minor revision number

OCCrev the returned revision of OpenCASCADE in use

Returns the version information for both EGADS and OpenCASCADE.

Open

```
icode = EG_open(ego *context);
```

context the returned Context Object

icode the integer return code

Opens and returns a Context object. This is required for the use of all EGADS (except for the above).

Close a Context

```
icode = EG_close(ego context);
```

context the Context Object to close

icode the integer return code

Cleans up and closes the Context.

Delete Object

```
icode = EG_deleteObject(ego object);
```

object the Object to delete

icode the integer return code

Deletes an Object (if possible). A positive return indicates that the Object is still referenced by this number of other Objects and has not been removed from the Context. If the Object is the Context then all Geometry/Topology Objects in the Context are deleted except those attached to Body or Model Objects.

Read Geometric data from a File

```
icode = EG_loadModel(ego context, int bitFlag, const char *name,  
                    ego *model);
```

context the Context Object to receive the geometry

bitFlag Options (additive):

- 1 Don't split closed and periodic entities
- 2 Split to maintain at least C^1 in BSPLINES
- 4 Don't maintain Units on STEP/IGES read (always millimeters)
- 8 Try to merge Edges and Faces (with same geometry)
- 16 Load unattached Edges as WireBodies (stp/step & igs/iges)

name path of file to load (with extension – case insensitive):

- igs/iges IGES file
- stp/step STEP file
- brep native OpenCASCADE file
- egads native file format with persistent Attributes (splits ignored)

model the returned Model Object that was read

icode the integer return code

Loads and returns a Model Object from disk and puts it in the Context.

Writes the Model to a File

```
icode = EG_saveModel(const ego object, const char *name);
```

object the Model Object to write

name path of file to write, type based on extension (case insensitive):

igs/iges IGES file

stp/step STEP file

brep a native OpenCASCADE file

egads a native file format with persistent Attributes and the ability to write EBody and Tessellation data

icode the integer return code

Writes the BReps (with optional Tessellation and EBody Objects) contained in the Model to disk. Only writes BRep data for anything but EGADS output.

Note: **object** can be a single Body for convenience

In the *exercise/session02* directory:

- Examine the Makefile (or NMakefile on Windows). Notice the library(s) included.
- Build the executable and run it. The output should look like:

```
Using EGADS 1.29 with OpenCASCADE 7.8.1
```

```
Number of Bodies = 2
```

```
Body 0: Name = capsLength String = cm
```

```
  Tessellation 0 npts = 1306 (Solid)
```

```
  Volumes = 7.280172e+00 7.296560e+00
```

```
Body 1: Name = capsLength String = cm
```

```
  Tessellation 0 npts = 2263 (Solid)
```

```
  Volumes = 3.454432e+01 3.464152e+01
```

```
EGADS Info: 0 Objects, 0 Reference in Use (of 247) at Close!
```

- Examine the EGADS API calls. Does the argument passing make sense?