



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

Bob Haimes

[bob@geocentrictech.com](mailto:bob@geocentrictech.com) or [haimes@mit.edu](mailto:haimes@mit.edu)

Geocentric Technologies LLC

## Create a simple Solid Body

```
icode = EG_makeSolidBody(ego context, int stype, const double *data,  
                        ego *body);
```

**context** the Context Object

**stype** one of: BOX, SPHERE, CONE, CYLINDER, TORUS

**data** length and fill depends on **stype**:

|          |   |                                                                                      |
|----------|---|--------------------------------------------------------------------------------------|
| BOX      | 6 | $[x, y, z]$ then $[dx, dy, dz]$ for the size of box                                  |
| SPHERE   | 4 | $[x, y, z]$ of center then the radius                                                |
| CONE     | 7 | apex $[x, y, z]$ , base center $[x, y, z]$ , then the radius                         |
| CYLINDER | 7 | 2 axis points and the radius                                                         |
| TORUS    | 8 | $[x, y, z]$ of center, direction of rotation, then the major radius and minor radius |

**body** the resultant Solid Body Object

**icode** the integer return code

## Queries the Objects in a Body

```
icode = EG_getBodyTopos (const ego body, ego ref, int oclass,  
                        int *ntopo, ego **topos);
```

**body** the Body Object

**ref** reference Topology Object or **NULL**. Sets the context for the returned Objects (i.e., all objects of the class **oclass** in the tree looking towards that class from **ref**) **NULL** starts from the **body** (for example all Nodes in the Body)

**oclass** is NODE, EDGE, LOOP, FACE or SHELL – must not be the same class as **ref** for EBODY can be EEDGE, ELOOPX, EFACE, ESHELL or the above

**ntopo** the returned number of Topology Objects

**topos** is a returned pointer to the vector of Objects, it is possible that an individual Object may be **NULL** (*freeable*)

Note: the argument can be **NULL** so the Objects are not filled

**icode** the integer return code

This allows for the traversal of the Topology *tree* by jumping levels and/or looking up the hierarchy.

## Get the index of the Object in a Body

```
index = EG_indexBodyTopo(const ego body, const ego obj);
```

**body** the Body Object

**obj** is the Topology Object in the Body

**index** the index (bias 1) or the integer return code (negative)

## Get the Object in a Body by index

```
icode = EG_objectBodyTopo(const ego body, int oclass, int index,  
                           ego *obj);
```

**body** the Body Object

**oclass** the Topology Object class

**index** the index (bias 1) of the entity requested

**obj** is the returned Topology Object from the Body

**icode** the integer return code

## Return the Bounding Box info

```
icode = EG_getBoundingBox(const ego object, double *bbox);
```

**object** any topological object

**bbox** 6 **double**s filled reflecting  $[x, y, z]_{min}$  and  $[x, y, z]_{max}$

**icode** the integer return code

Computes the smallest Cartesian bounding box surrounding the **object**.

## Returns the Mass Properties

```
icode = EG_getMassProperties(const ego object, double *props);
```

**object** can be EDGE, LOOP, FACE, SHELL, BODY or *Effective Topology* counterpart

**props** 14 **double**s filled reflecting Volume, Area (or Length), Center of Gravity (3) and the inertia matrix at CG (9)

**icode** the integer return code

Computes and returns the physical and inertial properties of a Topology Object.

## Memory Functions

These functions need to be used instead of the C/C++ variants for persistent memory due to the need to allocate/free from the same DLL under Windows.

```
EG_free(void *ptr);
```

```
void *ptr = EG_alloc(size_t nbytes);
```

```
void *ptr = EG_calloc(size_t nele, size_t size);
```

```
void *ptr = EG_reall(void *pointer, size_t nbytes);
```

```
char *str = EG_strdup(const char *string);
```

## Copy and optionally Transform an Object

```
icode = EG_copyObject(const ego object, void *other, ego *newObj);
```

- object** the Object to copy
- other** Transformation Object, Body Object, **NULL** for a strict copy, or a vector of **doubles**
- newObj** The resultant new Object
- icode** the integer return code

Creates a new EGADS Object by copying and optionally transforming the input object. A Tessellation or PCurve Object cannot be transformed. For a Tessellation Object, **other** can be a vector of displacements that is 3 times the number of vertices of **doubles** in length to *morph* the tessellation. Also, if **object** is a Tessellation Object or an EBody Object and **other** is a Body Object, the existing Object is copied but associated with the Body specified (not the original referenced object). Note that **other** is not checked if it is compatible with the original referenced Body.

If **other** is a Context, then **object** is copied to this target Context. This is useful in multithreaded settings.

## Get the Context

```
icode = EG_getContext(ego object, ego *context);
```

- object** the queried Object
- context** the returned owning Context
- icode** the integer return code

- Attributes – metadata consisting of name/value pairs
  - Unique name – no spaces
  - A single type: Integer, Real, String, CSys, Pointer (not persistent)
  - A length (for Integers & Reals)
- Objects
  - Any (non-internal) Object can have multiple Attributes
  - Only Attributes on Topological Objects are copied and are persistent (saved)
- SBO & Intersection Functions
  - Unmodified Topological Objects maintain their Attributes
  - Face Attributes are carried through to the resultant fragments
  - All other Attributes may be lost
- CSys Attributes are modified through Transformations

## Add an Attribute to an Object

```
icode = EG_attributeAdd(ego object, const char *name, int type,  
                        int len, const int *ints, const double *reals,  
                        const char *string);
```

- object** the Object to attribute
- name** the name of the attribute
- type** the attribute type:  
ATTRINT, ATTRREAL, ATTRSTRING, ATTRCSYS or ATTRPTR
- len** the number of integers or reals (ignored for strings and pointers)
- ints** the integers for ATTRINT
- reals** the floating-point data for ATTRREAL or ATTRCSYS
- string** the character string pointer for ATTRSTRING or ATTRPTR types
- icode** the integer return code

### Notes:

- 1 Only the one appropriate attribute value (of **ints**, **reals** or **string**) is required.
- 2 If the **name** already exists the type and value(s) are overwritten.

## Delete an Attribute from an Object

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

**object** the Object  
**name** the name of the attribute to delete  
**icode** the integer return code

Deletes an attribute from the Object. If the name is **NULL** then all attributes are removed from this Object.

## The number of Object Attributes

```
icode = EG_attributeNum(ego object, int *nAttr);
```

**object** the Object  
**nAttr** the returned number of attributes attached to the Object  
**icode** the integer return code

Returns the number of attributes found with this object.

## Return an Attribute on an Object

```
icode = EG_attributeRet(ego object, const char *name, int *type,  
                       int *len, const int **ints,  
                       const double **reals, const char **string);
```

**object** the Object to query

**name** the name to query

**type** the type: ATTRINT, ATTRREAL, ATTRSTRING, ATTRCSYS or ATTRPTR

**len** the returned number of integers or reals

**ints** the returned pointer to integers for ATTRINT

**reals** the returned pointer to floating-point data for ATTRREAL or ATTRCSYS

**string** the returned pointer to a character string for ATTRSTRING or ATTRPTR types

**icode** the integer return code

### Notes:

- 1 Only the appropriate attribute value (of **ints**, **reals** or **string**) is returned.
- 2 The CSys (12 reals) is returned in **reals** after the **len** values.

## Get an Attribute on an Object

```
icode = EG_attributeGet(ego object, int index, const char **name,  
                        int *type, int *len, const int **ints,  
                        const double **reals, const char **string);
```

**object** the Object to query

**index** the index (1 to **nAttr** from EG\_attributeNum)

**name** the returned name

**type** the type: ATTRINT, ATTRREAL, ATTRSTRING, ATTRCSYS or ATTRPTR

**len** the returned number of integers or reals

**ints** the returned pointer to integers for ATTRINT

**reals** the returned pointer to floating-point data for ATTRREAL or ATTRCSYS

**string** the returned pointer to a character string for ATTRSTRING or ATTRPTR types

**icode** the integer return code

### Notes:

- 1 Only the appropriate attribute value (of **ints**, **reals** or **string**) is returned.
- 2 The CSys (12 reals) is returned in **reals** after the **len** values.

## Copy the Attributes from an Object to another

```
icode = EG_attributeDup(ego src, ego dst);
```

**src** the source Object

**dst** the Object to receive **src**'s attributes

**icode** the integer return code

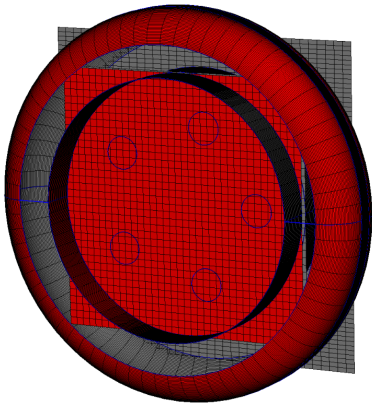
Deletes an attribute from the destination Object and then copies the source's attributes to the destination. ATTRPTR attributes copy the pointer, other types allocate new data and copy the contents of the source.

## Geometry

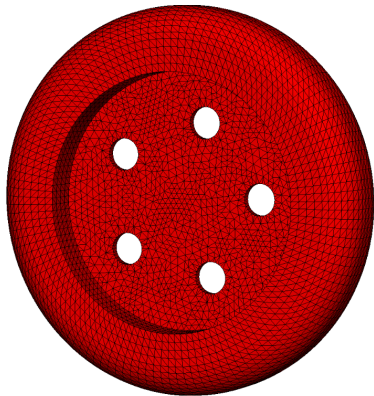
- Unconnected discretization of a range of the Object
  - Polyline for curves at constant  $t$  increments
  - Regular grid for surfaces at constant increments (isoclines)

## Body Topology

- Connected and trimmed tessellation including:
  - Polyline for Edges
  - Triangulation for Faces
  - Optional Quadrilateral Patching for Faces
- Ownership and Geometric Parameters for Vertices
- Adjustable parameters for side length and curvature (x2)
- Watertight
- Exposed per Face/Edge or Global indexing



from `$ESP_ROOT/bin/vGeom`



from `$ESP_ROOT/bin/vTess`

## Creates a Discrete Object from a Body

```
icode = EG_makeTessBody(ego body, double *parms, ego *tess);
```

- body** the input Body or closed EBody Object, may be any Body type
- parms** a set of 3 parameters that drive the Edge discretization and the Face triangulation:
  - parms[0]** – the maximum length of an Edge segment or triangle side (in physical space); a zero is no limit, and a negative value only tessellates Edges.
  - parms[1]** – 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.
  - parms[2]** – 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.
- tess** the returned resultant Tessellation of **body**
- icode** the integer return code

See the next page for attribute-based tessellation control.

## Tessellation control at the Topological level

- .tParams** this attribute can be placed on the Body, individual Faces or Edges which overrides **parms** locally (the minimums are used). This attribute must be ATTRREAL and have 3 values (as described in EG\_makeTessBody).
- .tParam** like the attribute **.tParams**, this attribute completely overrides **parms** locally (without using the minimum).
- .tPos** this ATTRREAL attribute on an Edge directly sets the *ts* for interior Edge positions.
- .rPos** this ATTRREAL attribute sets the relative spacing (in arc-length) for interior Edge positions.
- .nPos** this ATTRINT attribute sets the number of interior vertices (length is 1). The spacing is set equal in arc-length.
- .inserts** this ATTRREAL attribute (on a Face) specifies that these vertex  $[u, v]$  positions will be inserted into the tessellation. The length must be 2 times the number of inserts.
- .insert!** like the attribute **.inserts**, this specifies the  $[u, v]$  positions to be inserted, but after these inserts the Face tessellation terminates (i.e., no additional insertions are performed by the normal algorithm).

Note:

An ATTRINT attribute **.tPos** or **.rPos** of length 1 and containing a zero indicates no interior points.

## Gets the Edge discretization data

```
icode = EG_getTessEdge(const ego tess, int eIndex, int *len,  
                      const double **xyzs, const double **ts);
```

- tess** the input Body Tessellation Object
- eIndex** the Edge index (1 bias). The Edge Objects and number of Edges can be retrieved via `EG_getBodyTopos` and/or `EG_indexBodyTopo`. A minus index refers to the use of a mapped (+) Edge index from applying the functions `EG_mapBody` and `EG_mapTessBody`.
- len** the returned number of vertices in the Edge discretization
- xyzs** the returned pointer to the set of coordinate data –  $3*\text{len}$  in length
- ts** the returned pointer to the parameter values associated with each vertex –  $\text{len}$  in length
- icode** the integer return code

Note: DEGENERATE Edges return 2 vertices (both the same coordinates of the single Node) and the  $t$  range in **ts**. This Edge will not be referenced in the associated Face tessellation.

## Gets the Face triangulation data

```
icode = EG_getTessFace(const ego tess, int fIndex, int *len,
                      const double **xyz, const double **uv,
                      const int **ptype, const int **pindx, int *ntri,
                      const int **tris, const int **tric);
```

- tess** the input Body Tessellation Object
- fIndex** the Face index (1 bias) – Minus index refers to a mapped (+) Face index (if it exists).
- len** the returned number of vertices in the Face triangulation
- xyz** the returned pointer to the set of coordinate data – 3\***len** in length
- uv** the returned pointer to the parameters for each vertex – 2\***len** in length
- ptype** returned pointer to the vertex type (-1 - internal, 0 - Node, > 0 Edge) – **len** in length
- pindx** returned pointer to vertex index (-1 internal) – **len** in length
- ntri** returned number of triangles
- tris** returned pointer to triangle indices, 3 per triangle (1 bias) – 3\***ntri** in length  
orientation consistent with the Face's **mtype**
- tric** returned pointer to neighbor information, 3 per triangle looking at opposing side:  
triangle (1-ntri), negative is Edge index for an external side – 3\***ntri** in length
- icode** the integer return code

## Status of a Tessellation Object

```
icode = EG_statusTessBody(ego tess, ego *body, int *stat, int *npts);
```

- tess** the Tessellation Object to query
- body** the returned associated Body Object
- stat** the returned state of the tessellation: -1 – closed but warned, 0 – open, 1 – OK, 2 – displaced
- npts** the returned number of global points in the tessellation (0 – open)
- icode** the integer return code: EGADS\_SUCCESS – complete, EGADS\_OUTSIDE – still open

Note: Placing the attribute “.mixed” on **tess** before invoking this function allows for tri/quad (2 tris) tessellations. The type must be ATTRINT and the length is the number of Faces, where the values are the number of quads (triangle pairs) per Face. Single triangles are followed by triangle pairs for a Face with both triangle and quads.

|                        |       |
|------------------------|-------|
| Given quad 1 2 3 4 ==> | 4---3 |
| trias 1 2 3 and 1 3 4  | /     |
|                        | 1---2 |

## Global Lookup

```
icode = EG_localToGlobal(const ego tess, int ind, int locl, int *gbl);
```

**tess** the closed Tessellation Object

**ind** the topological index (1 bias) – 0 Node, (-) Edge, (+) Face

**locl** the local (or Node) index

**gbl** the returned global vertex index

**icode** the integer return code

## Gets the vertex type and index

```
icode = EG_getGlobal(const ego tess, int global, int *pytpe,  
                    int *pindex, double *xyz);
```

**tess** the closed Tessellation Object

**global** the global index (1 bias)

**pytpe** the point type (-) Face local index, (0) Node, (+) Edge local index

**pindex** the point topological index (1 bias)

**xyz** the filled (3 in length) coordinates at this global index (can be **NULL**)

**icode** the integer return code

In the *exercise/session03* directory:

- Modify `myExample.c` to print all attributes using `EG_getBodyTopos` for all of the **FACES**, **EDGES** and **NODES** in each **Body**.  
Note that you must *free* the vectors of **Objects**, but not the **Objects** themselves (which are owned by the **Body**).
- Modify `myExample.c` to traverse the **BRep Topology** from **Model** to **Nodes** using `EG_getTopology` and output all attributes attached to each **Topological entity**.  
Why do you see attributes repeated?