EGADS:

Engineering **G**eometry **A**ircraft **D**esign **S**ystem

Bob Haimes
haimes@mit.edu
Aerospace Computational Design Lab
Department of Aeronautics & Astronautics
Massachusetts Institute of Technology

Revision 1.17 – January 2020

Outline

- **Overview**
- **Objects**
  - Geometry
  - Topology
  - Tessellation
  - Others
- **API**
  - Utility & IO Functions
  - Attribution
  - Geometry
  - Topology
  - Tessellation
  - High-Level Functions

Overview

Provide a "bottom up" and/or Constructive Solid Geometry foundation for building Aircraft

- **Built upon OpenCASCADE**
  - Open Source solid modeling geometry kernel
  - Support for manifold and non-manifold data
  - Reading & writing IGES, STEP and native formats
  - C++ with ~17,000 methods!
- **Open Source** (LGPL v2.1)
- **C/C++ and FORTRAN Interfaces**
  - Single API with minor variations for FORTRAN
  - Always returns an integer code (success/error condition)
  - Requires C pointer access in FORTRAN
    - Cray-pointer construct
    - C-pointers (2003 extension to FORTRAN 90)
    - Both supported by Intel FORTRAN and **gfortran**
    - API contains memory functions

Overview

- **System Support (now 64 bit only):**
  - **Mac OSX** with **gcc**, **clang**, **ifort** and/or **gfortran**
  - **LINUX** with **gcc**, **ifort** and/or **gfortran**
  - **Windows** with Microsoft Visual Studio C++ and **ifort**
  - No globals and thread-safe
  - Various levels of output (0-none, through 3-debug)
  - Written in C and C++

- **EGADS Objects**
  - Treated as "blind" pointers – an ego
    - Can access internals in C/C++
    - Context object hold *global* information
  - egos are INTEGER*8 variables in FORTRAN
    - Allows for same source code regardless of size of pointer
    - Requires "freeing" of internal lists of objects (not required in C/C++)

Overview

- **Context and Thread**
  - When a Context Object is created (**`EG_open`**) the calling thread ID is saved in the Context
  - Any construction functions or functions that change the attribute storage must be done from the thread stored in the Context
  - The Context's thread may be modified by invoking **`EG_updateThread`** called from the new thread

- **MultiThreading**
  - After a thread is spawned, it can call **`EG_open`** to setup a Context to use with the thread
    - Will work with native threads, **ESP**'s EMP package or **OpenMP**
  - Use **`EG_copyObject`** to copy an object from its owning Context to the target Context specified in the 2nd argument
  - See the sample **`multiContext.c`** for an example using EMP

Objects

- **C Structure Definition:**

```
typedef struct egObject {
   int    magicnumber;        /* must be set to validate the object */
   short  oclass;             /* object class */
   short  mtype;              /* object member type */
   void  *attrs;              /* attributes or reference */
   void  *blind;              /* blind pointer to OCC 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**
  - Holds 'globals' including output level
  - Start of dual threaded-list of active egos
  - Pool of deleted objects
  - Owning thread ID

Objects – Attribution

- **Attributes**
  - Are identified by a **name** (character string with no spaces or other special characters)
  - Each named attribute has a single **type**:
    - Integer
    - Real (double precision)
    - String (can have spaces and other special characters)
    - CSys – Coordinate System (uses the Real storage)
    - Ptr – Supplied pointer (not persistent and the programmer is responsible for memory management, i.e. freeing the storage). Uses the String pointer.
  - And a **length** (for Integer, Real and CSys types)
- **Objects**
  - Any Object (except for REFERENCE) may have multiple Attributes
  - Only Attributes on Topological Objects are ***copied*** (except for Pointers).
  - Only Attributes on Topological Objects are ***persistent*** (except for Pointer Types) – and this is available only for ".egads" file IO.
- **SBO and Intersection Functions**
  - Attributes on Faces will be carried through to the resultant fragments after intersections (except for Pointer types).
  - Unmodified Topology maintains their attributes (except for Pointers).

Objects – Attribution

- **Some operations return more complete associations**
  - Attributes on Faces are always copied from the source regardless of the Function* (an exact copy, trimmed or split)
  - These return a list of Face mappings for each Face in the result:
    - **`EG_filletBody`**
    - **`EG_chamferBody`**
    - **`EG_hollowBody`**
  - The list contains an operation and an index to the source object:

Operation	
NODEOFF (1)	The Face is the result of a Node – the index is that of the Node in the source Body
EDGEOFF (2)	The Face is the result of an Edge – the index is the Edge index (see **`EG_indexBodyTopo`**)
FACEDUP* (3)	The Face is an exact copy of the source
FACECUT* (4)	The Face has been trimmed or split from the source
FACEOFF (5)	The Face is offset from the source Face

Objects – Attribution

- **Coordinate Systems – ATTRCSYS**
  - Input Reals must be one of:
    - Any Object may have 9 values
      - position°, first direction°, second direction°
    - FACE/SURFACE can also have 6 or 3 values
      - u, v, flip, second direction° (first direction is flip*normal) --or--
      - u,v, *idir*: 1 − udir, 2 − vdir, 3 − -udir, 4 − -vdir
        first direction is the normal, second is set by *idir*
        if *idir* is negated then the normal direction is flipped
    - EDGE/CURVE can have 5 values
      - t, flip, second direction° (first direction is flip*tangent)
    - NODE can have 6 values
      - first direction°, second direction°
  - Output as position and 3 orthonormal directions − 12 doubles returned after the input values for **`EG_attributeGet`** and **`EG_attributeRet`**
  - Notes:
    - third direction is implied by first ✖ second
    - ° Transformed when object has been transformed
    - The actual number of doubles is the length above + 12

Objects – Geometry

- **PCURVE – Parameter Curves**
  - 2D Curves in the Parametric space *[u,v]* of a Surface
  - Single "running" parameter *t*
  - *[u,v] = f(t)*
- **CURVE**
  - 3D Curves
  - Single "running" parameter *t*
  - *[x,y,z] = f(t)*
- **SURFACE**
  - 3D Surfaces of 2 parameters *[u,v]*
  - *[x,y,z] = f(u,v)*

Objects – PCURVE/CURVE

- **LINE**

	Curve *(6)*	PCurve *(4)*
Location	*[x,y,z]*	*[u,v]*
Direction	*[dx,dy,dz]*	*[du,dv]*

- **CIRCLE**

	Curve *(10)*	PCurve *(7)*
Center	*[x,y,z]*	*[u,v]*
Xaxis	*[dx1,dx2,dx3]*	*[dx1,dx2]*
Yaxis	*[dy1,dy2,dy3]*	*[dy1,dy2]*
Radius		

note: **Xaxis** and **Yaxis** should be orthogonal

Objects – PCURVE/CURVE

- **ELLIPSE**

	Curve *(11)*	PCurve *(8)*
Location	*[x,y,z]*	*[u,v]*
Xaxis	*[dx1,dx2,dx3]*	*[dx1,dx2]*
Yaxis	*[dy1,dy2,dy3]*	*[dy1,dy2]*
MajorRadius		
MinorRadius		

note: **Xaxis** and **Yaxis** should be orthogonal

- **PARABOLA**

	Curve *(10)*	PCurve *(7)*
Location	*[x,y,z]*	*[u,v]*
Xaxis	*[dx1,dx2,dx3]*	*[dx1,dx2]*
Yaxis	*[dy1,dy2,dy3]*	*[dy1,dy2]*
Focus		

note: **Xaxis** and **Yaxis** should be orthogonal

Objects – PCURVE/CURVE

- **HYPERBOLA**

	Curve *(11)*	PCurve *(8)*
Location	*[x,y,z]*	*[u,v]*
Xaxis	*[dx1,dx2,dx3]*	*[dx1,dx2]*
Yaxis	*[dy1,dy2,dy3]*	*[dy1,dy2]*
MajorRadius		
MinorRadius		

note: **Xaxis** and **Yaxis** should be orthogonal

- **TRIMMED** (has Reference Geometry)
  - 2 in length for both Curve types (***t*-start** & ***t*-end**)

- **OFFSET** (has Reference Geometry)

	Curve *(4)*	PCurve *(1)*
Direction	*[dx,dy,dz]*	-
Offset		

Objects – PCURVE/CURVE

- **BEZIER** (3 integer header):

BitFlag	2 - rational, 4 - periodic	
Degree	(not used on input, is **nCP**-1 up to 25)	
nCP		
	Curve	PCurve
ControlPts	*3*nCP*	*2*nCP*
Weights	*nCP*	*nCP*

note: **Weights** only if rational

- **BSPLINE** (4 integer header):

BitFlag	2 - rational, 4 - periodic	
Degree		
nCP		
nKnots		
	Curve	PCurve
Knots	*nKnots*	*nKnots*
ControlPts	*3*nCP*	*2*nCP*
Weights	*nCP*	*nCP*

note: **Weights** only if rational

Objects – SURFACE

- **PLANE** (9 doubles in length):

  Location *[x,y,z]*
  Xaxis *[dx1,dx2,dx3]*
  Yaxis *[dy1,dy2,dy3]*
  note: **Xaxis** and **Yaxis** should be orthogonal

- **SPHERICAL** (10 doubles in length):

  Center *[x,y,z]*
  Xaxis *[dx1,dx2,dx3]*
  Yaxis *[dy1,dy2,dy3]*
  Radius

  notes: **Xaxis** and **Yaxis** should be orthogonal
  negative **Radius** indicates a left-handed coordinate system

Objects – SURFACE

- **CONICAL** (14 doubles in length):

  Location *[x,y,z]*
  Xaxis *[dx1,dx2,dx3]*
  Yaxis *[dy1,dy2,dy3]*
  Direction *[dz1,dz2,dz3]* rotation axis *(may be LeftH)*
  Angle
  Radius
  note: **Xaxis**, **Yaxis** and **Direction** should be orthogonal

- **CYLINDRICAL** (13 doubles in length):

  Center *[x,y,z]*
  Xaxis *[dx1,dx2,dx3]*
  Yaxis *[dy1,dy2,dy3]*
  Direction *[dz1,dz2,dz3]* rotation axis *(may be LeftH)*
  Radius
  note: **Xaxis** and **Yaxis** should be orthogonal

Objects – SURFACE

- **TOROIDAL** (14 doubles in length):

  Location *[x,y,z]*
  Xaxis *[dx1,dx2,dx3]*
  Yaxis *[dy1,dy2,dy3]*
  Direction *[dz1,dz2,dz3]* rotation axis *(may be LeftH)*
  MajorRadius
  MinorRadius
  note: **Xaxis**, **Yaxis** and **Direction** should be orthogonal

- **REVOLUTION** (6 doubles in length):

  Center *[x,y,z]*
  Direction *[dx,dy,dz]*

- **EXTRUSION** (3 doubles in length):

  Direction *[dx,dy,dz]*

Objects – SURFACE

- **BEZIER** (5 integer header):

BitFlag	2 - rational, 4 - uPeriodic, 8 - vPeriodic
uDegree	(not used on input, is **nCPu**-1 up to 25)
nCPu	
vDegree	(not used on input, is **nCPv**-1 up to 25)
nCPv	
	Data Packed:
ControlPts	*3*nCPu*nCPv*
Weights	*nCPu*nCPv*

note: **Weights** only if rational

- **TRIMMED** (has Reference Geometry)
  - 4 in length (***u*-start**, ***u*-end**, ***v*-start** & ***v*-end**)
- **OFFSET** (has Reference Geometry)
  - 1 in length – ***offset distance***

Objects – SURFACE

- **BSPLINE** (7 integer header):

BitFlag	2 - rational, 4 - uPeriodic, 8 - vPeriodic
uDegree	
nCPu	
nUKnots	
vDegree	
nCPv	
nVKnots	
	Data Packed:
uKnots	*nUKnots*
vKnots	*nVKnots*
ControlPts	*3*nCPu*nCPv*
Weights	*nCPu*nCPv*

note: **Weights** only if rational

Objects – Topology

EGADS Topological Entity	OpenCASCADE term	Geometric Entities
Model	**Compound Shape**	
Body	**Solid (or lesser shape)**	
Shell		
Face		**surface**
Loop	**Wire**	* see note below
Edge		**curve**
Node	**Vertex**	

- Topological entities have children (entities lower on the table) except for **Nodes**

* **Loops** may be geometry free or have associated **PCurves** (one for each **Edge**) and the **surface** where the **PCurves** reside

Objects – Topology

- **NODE**
  - Contains *[x,y,z]*
- **EDGE**
  - Has a 3D **CURVE** (if not **DEGENERATE**)
  - Has a ***t*** range (***t***min to ***t***max, where ***t***min < ***t***max)
  - The positive orientation is going from ***t***min to ***t***max
  - Has a **NODE** for ***t***min and for ***t***max
  - Can be **ONENODE** (closed or periodic), **TWONODE**, or **DEGENERATE** (which has a single **NODE** and a valid range which will be used for the associated **PCurve**)

Objects – Topology

- **LOOP** (without a reference **SURFACE**)
  - Free standing collection of **EDGE**s that can be used in a non-manifold setting (for example in WireBodies)
  - Collections of **EDGE**s associated with a **PLANE** which does not require **PCurves** in OpenCASCADE
  - An ordered collection of **EDGE** objects with associated senses that define the connected Wire
  - Segregates space by maintaining *material* to the left of the running **LOOP** (or traversed right-handed pointing out of the intended volume)
  - No **EDGE**s should be **DEGENERATE**
  - Can be **OPEN** or **CLOSED** (comes back on itself)

Objects – Topology

- **LOOP** (with a reference **SURFACE**)
  - Collections of **EDGE**s (like without a **SURFACE**) followed by a corresponding collection of **PCurves** that define the *[u,v]* trimming on the **SURFACE**
  - **DEGENERATE EDGE**s 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)
  - An **EDGE** may be found in a **LOOP** twice (with opposite senses) and with different **PCurves**. For example a closed cylindrical surface at the seam -- one **PCurve** would represent the beginning of the period where the other is the end of the periodic range.

Objects – Topology

- **FACE**
  - A **SURFACE** bounded by one or more **LOOP**s with associated senses
  - Only one outer **LOOP** (sense = 1) and any number of inner **LOOP**s (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 **LOOP**s must be **CLOSED**
  - If the **SURFACE** is a **PLANE**, the **LOOP**(s) must not contain any reference geometry
  - If the **SURFACE** is not a **PLANE** then the **LOOP**'s reference Object must match that of the **FACE**
  - The orientation of the **FACE** is either **SFORWARD** (where the **SURFACE**'s natural normal (*U*x*V*) matches the **FACE**) or **SREVERSE** when the orientations are apposed. 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)

Objects – Topology

- **SHELL**
  - A collection of one of more connected **FACE**s that (if **CLOSED**) segregates regions of 3-Space
  - All **FACE**s must be properly oriented
  - **SHELL**s can be either **OPEN** or **CLOSED**
  - Non-manifold **SHELL**s can have more than 2 **FACE**s sharing an **EDGE** (**OPEN** in this case)
- **SOLIDBODY**
  - A manifold collection of one or more **CLOSED SHELL**s with associated senses
  - There may be only one outer **SHELL** (sense = 1) and any number of inner **SHELL**s (sense = -1)

Objects – Topology

- **BODY** (including **SOLIDBODY**)
  - Container used to aggregate Topology
  - Connected but non-manifold at the **MODEL** level
  - A **WIREBODY** contains a single **LOOP**
  - A **FACEBODY** contains a single **FACE**
  - A **SHEETBODY** contains one or more **SHELL**s which can be either non-manifold or manifold (though usually manifold bodies of this type are promoted to **SOLIDBODY**s)
- **MODEL**
  - A collection of **BODIES**
  - Can be treated like Assemblies
  - This is Read and Written by **EGADS**

Objects – Tessellation

Discrete representation of another Object

- **Geometry**
  - Unconnected discretization of a range of the Object
    - PolyLine for **CURVE**s at constant ***t*** increments
    - Regular Grid for **SURFACE**s at constant increments

- **Body Topology**
  - Connected and trimmed tessellation including:
    - PolyLine for **EDGE**s
    - Triangulation for **FACE**s
    - Optional Quadrilateral Patching for **FACE**s
  - Ownership and Geometric Parameters for Vertices
  - Adjustable Parameters for side length and curvature
  - Watertight

Objects – Tessellation

Control of the use of Quadrilateral Templates

- Automatic with triangulation scheme
- Attempts to Isolate 3 or 4 "sides"
  - Only single **LOOP**s
  - **FACE**s with more than 4 **EDGE**s are analyzed to see is multiple **EDGE**s can be treated as a single "side"
- Point counts on sides (based on **EDGE** Tessellation) are used:
  - TFI if opposites are equal
  - Templates otherwise
- Defeated/modified with **BODY** or **FACE** attribute ".qParams"
  - If ATTRSTRING – turn off quadding templates
  - If ATTRREAL (3 in length):
    1. EDGE matching expressed as the deviation from alignment [default: *0.05*]
    2. Maximum quad *side* ratio point count to allow [default: *3.0*]
    3. Number of smoothing iterations [default: *0.0*]

Objects – Tessellation

Watertight Quadrilateral FACE Treatment

- Manual
- Requires Existing Topologic Tessellation
- Must be able to Isolate 4 "sides"
  - Only single **LOOP**s
  - **FACE**s with more than 4 **EDGE**s are analyzed to see is multiple **EDGE**s can be treated as a single "side"
  - Currently no DEGENERATE **EDGE**s
- Point counts on sides (based on **EDGE** Tessellation) are used:
  - TFI if opposites are equal
  - Templates otherwise
- **EDGE** Tessellation Adjustment Functions
  - When point counts don't allow for Quadding

Objects – Tessellation

Using Tessellations for Finite-Differences

- Useful for Parametric Sensitivities in a parameter driven build system
- Requires the same Topologic structure between Bodies (may need a *mapping*)
- BSpline **SURFACE**s (on mapped **FACE**s) must have the same knot sequences (note: the knots define the *[u,v]* parametrization).
- This is accomplished by 2 EGADS functions:
  - **`EG_mapBody`** – sets up the mapping if required
  - **`EG_mapTessBody`** – builds the tessellation from a source
- These functions respond to mapped indices:
  - **`EG_getTessEdge`**
  - **`EG_getTessFace`**
  - **`EG_locateTessBody`**

Objects – Others

- **TRANSFORM**
  - Used when copying Objects to change the root position and orientation
- **REFERENCE**
  - Allows of the management of Objects that refer to other Objects (so that deletion does not invalidate the data)
  - This is an internal Object and is not usually seen by the EGADS programmer.

Objects – Lifetime & Scope

- **BODY**
  - When made, copies of all referenced objects are created and stored
- **MODEL**
  - A **BODY** can be included in only one **MODEL** (you will get a "reference error" if violated)
  - Copy the **BODY** if it is needed in a second **MODEL**
- **Others**
  - Unconnected (at the **BODY**-level) Geometric & Topologic Objects can be deleted *en masse* by invoking **`EG_deleteObject`** on the **CONTEXT**

API – Utility & IO Functions

- **open**

```
icode = EG_open(ego *context)
icode = IG_open(I*8  context)
```

Opens and returns a **CONTEXT** object. Note that the **Context** is the beginning of the threaded list of objects.

- **updateThread**

```
icode = EG_updateThread(ego context)
icode = IG_updateThread(I*8 context)
```

Resets the Context's owning thread to the one calling this function.

- **deleteObject**

```
icode = EG_deleteObject(ego object)
icode = IG_deleteObject(I*8 object)
```

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 objects in the context are deleted except those attached to **BODY** or **MODEL** objects.

API – Utility & IO Functions

- **getContext**

```
icode = EG_getContext(ego object, ego *context)
icode = IG_getContext(I*8 object, I*8  context)
```

  Returns the **CONTEXT** given an object

- **setOutLevel**

```
icode = EG_setOutLevel(ego context, int outLevel)
icode = IG_setOutLevel(I*8 context, I*4 outLevel)
```

  Sets the EGADS verbose level (0-silent to 3-debug), The default is 1. Success returns the old output level.

- **close**

```
icode = EG_close(ego context)
icode = IG_close(I*8 context)
```

  Cleans up and closes the **CONTEXT**

- **revision**

```
      EG_revision(int *major, int *minor, char **OCCrev)
call IG_revision(I*4  major, I*4  minor, C**     OCCrev)
```

  Returns the version information for EGADS and OpenCASCADE

API – Utility & IO Functions

- **loadModel**

```
icode = EG_loadModel(ego context, int flags,
                     char *name,  ego *model)
icode = IG_loadModel(I*8 context, I*4 flags,
                     C** name,    I*8 model)
```

Loads and returns a **MODEL** object from disk and put it in the **CONTEXT**.

Flags (additive):

1 - Don't split closed and periodic entities
2 - Split to maintain at least C1 in BSPLINEs
4 - Don't try maintaining Units on STEP read (always millimeters)
8 - Try to merge Edges and Faces (with same geometry)

name: Load by extension

- igs/iges
- stp/step
- brep (for native OpenCASCADE files)
- egads (for native files with persistent Attributes, splits ignored)

- **saveModel**

```
icode = EG_saveModel(ego model, char *name)
icode = IG_saveModel(I*8 model, C**   name)
```

Saves the **MODEL** to disk based on the filename extension.

API – Utility & IO Functions

- **getTransformation**

```
icode = EG_getTransformation(ego oform, double *xform)
icode = IG_getTransformation(I*8 oform, R*8     xform)
```

Returns the transformation information. This appears like is a column-major matrix that is 4 columns by 3 rows and could be thought of as [3][4] in C (though is flat) and in FORTRAN dimensioned as (4,3).

oform	the transformation object
xform	a vector of double precision reals at least 12 in length

- **makeTransform**

```
icode = EG_makeTransform(ego context, double *xform,
                         ego *oform)
icode = IG_makeTransform(I*8 context, R*8     xform,
                         I*8  oform)
```

Creates a **TRANSFORM** object from the 12 values. The rotation portion [3][3] must be "scaled" orthonormal (orthogonal with a single scale).

API – Utility & IO Functions

- **copyObject**

```
icode = EG_copyObject(ego object, void *other,
                      ego *newObject)
icode = IG_copyObject(I*8 object, I*8   other,
                      I*8  newObject)
```

Creates a new EGADS object by copying and transforming the input object.

object	the input object (3D geometry, topology or tessellation)
other	the transformation or context object (an ego) -- NULL for a strict copy can be a displacement vector for TESSELLATION Objects only (number of global indices by 3 doubles in length)
newObject	the resultant new object

- **flipObject**

```
icode = EG_flipObject(ego object, ego *newObject)
icode = IG_flipObject(I*8 object, I*8  newObject)
```

Creates a new EGADS object by copying and reversing the input object.

object	the input object: 3D geometry (flip the parameterization) or topology (reverse the sense). Not for NODE, BODY or MODEL. SURFACEs reverse only the *u* parameter.
newObject	the resultant new flipped object

API – Utility & IO Functions

- **getInfo**

```
icode = EG_getInfo(ego object,  int *oclass, int *mtype,
                   ego *topRef, ego *prev,   ego *next)
icode = IG_getInfo(I*8 object,  I*4 oclass,  I*4 mtype,
                   I*8 topRef,  I*8 prev,    I*8 next)
```

Returns information about the object:

oclass	CONTEXT, TRANSFORM, TESSELLATION, REFERENCE, PCURVE, CURVE, SURFACE, NODE, EDGE, LOOP, FACE, SHELL, BODY, MODEL
mtype	PCURVE/CURVE LINE, CIRCLE, ELLIPSE, PARABOLA, HYPERBOLA, TRIMMED, BEZIER, BSPLINE, OFFSET SURFACE PLANE, SPHERICAL, CYLINDRICAL, REVOLUTION, TOROIDAL, TRIMMED, BEZIER, BSPLINE, OFFSET, CONICAL, EXTRUSION EDGE is TWONODE, ONENODE or DEGENERATE LOOP is OPEN or CLOSED FACE is either SFORWARD or SREVERSE SHELL is OPEN or CLOSED BODY is either WIREBODY, FACEBODY, SHEETBODY or SOLIDBODY
topRef	is the top level BODY/MODEL that owns the object or context (if top)
prev	is the previous object in the threaded list (NULL at CONTEXT)
next	is the next object in the list (NULL is the end of the list)

API – Utility & IO Functions

- **free**

```
      EG_free(void *ptr)
call IG_free(cptr  ptr)
```

  Used to free up a pointer returned from EGADS if marked as "freeable"

- **alloc** -- Allocates a block of memory

```
void * = EG_alloc(I*4 nbytes)
icode  = IG_alloc(I*4 nbytes, CPTR ptr)
```

- **calloc** -- Allocates a zero fills a block of memory

```
void * = EG_calloc(I*4 nele, I*4 size, CPTR ptr)
icode  = IG_calloc(I*4 nele, I*4 size, CPTR ptr)
```

- **reall** -- Reallocates a block of memory

```
void * = EG_reall(CPTR ptr, I*4 nbytes)
icode  = IG_reall(CPTR ptr, I*4 nbytes)
```

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

API – Attribution

- **attributeAdd**

```
icode = EG_attributeAdd(ego object, char *name, int atype,
                        int len, int *ints, double *reals,
                        char *string)
icode = IG_attributeAdd(I*8 object, C**   name, I*4 atype,
                        I*4 len, I*4  ints, R*8     reals,
                        C**   string)
```

Adds an attribute to the object. If an attribute exists with the name it is overwritten with the new information.

object	the object
name	the name of the attribute. Must not contain a space or other special characters
atype	must be either: ATTRINT for integers ATTRREAL for double precision ATTRSTRING for a character string ATTRCSYS for a coordinate system (use **`reals`** for input) ATTRPTR for a user managed pointer (use **`string`** for input)
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

Note: Only the appropriate one (of ints, reals or string) is required

API – Attribution

- **attributeDel**

```
icode = EG_attributeDel(ego object, char *name)
icode = IG_attributeDel(I*8 object, C**   name)
```

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

object	the object
name	the name of the attribute. FORTRAN can use a string containing just space(s) to indicate NULL

- **attributeNum**

```
icode = EG_attributeNum(ego object, int *nattr)
icode = IG_attributeNum(I*8 object, I*4  nattr)
```

Returns the number of attributes found with this object.

object	the object
nattr	the number of attributes

API – Attribution

- **attributeGet**

```
icode = EG_attributeGet(ego object, int index, char **name,
                        int *atype, int *len,  int **pints,
                        double **preals, char **string)
icode = IG_attributeGet(I*8 object, I*4  index, C**   name,
                        I*4 atype,  I*4   len,  CPTR pints,
                        R*8      preals,  C**   string)
```

Retrieves a specific attribute from the object.

object	the object
index	the index (1 to num from **`attributeNum`**)
name	the returned name of the attribute
atype	the returned type: ATTRINT, ATTRREAL, ATTRSTRING, ATTRCSYS, ATTRPTR
len	the returned length for integers or reals
pints	a pointer to integer(s) for ATTRINT
preals	a pointer to the floating point data for ATTRREAL or ATTRCSYS
string	the returned pointer for ATTRSTRING or ATTRPTR types

Notes: (1) Only the appropriate one (of pints, preals or string) is returned
(2) Care must be taken with name and string in FORTRAN not to overstep the declared CHARACTER length
(3) The CSys (12 reals) is returned in preals after the len values

API – Attribution

- **attributeRet**

```
icode = EG_attributeRet(ego object, char *name, int *atype,
                        int *len, int **pints, double **preals,
                        char **string)
icode = IG_attributeRet(I*8 object, C**   name, I*4  atype,
                        I*4  len, CPTR  pints,  R*8     preals,
                        C**    string)
```

Retrieves an attribute by name from the object.

object	the object
name	the name of the attribute to return
atype	the returned type: ATTRINT, ATTRREAL, ATTRSTRING, ATTRCSYS, ATTRPTR
len	the returned length for integers or reals
pints	a pointer to integer(s) for ATTRINT
preals	a pointer to the floating point data for ATTRREAL or ATTRCSYS
string	the returned pointer for ATTRSTRING or ATTRPTR types

Notes: (1) Only the appropriate one (of pints, preals or string) is returned
(2) Care must be taken with the string variable in FORTRAN not to overstep the declared CHARACTER length
(3) The CSys (12 reals) is returned in preals after the len values

API – Attribution

- **attributeDup**

```
icode = EG_attributeDup(ego src, ego dst)
icode = IG_attributeDup(I*8 src, I*8 dst)
```

Removes all attributes from the destination object, then copies the attributes from the source.

src	the source object
dst	the destination object

Note that attributes that are ATTRPTR types copy the pointer, others allocate new data and copy the contents of the source.

API – Geometry

- **getGeometry**

```
icode = EG_getGeometry(ego object, int *oclass, int *mtype,
                       ego *rGeom, int **pinfo, double **prv)
icode = IG_getGeometry(I*8 object, I*4 oclass,  I*4 mtype,
                       I*8 rGeom,  CPTR pinfo,  CPTR prv)
```

Returns information about the geometric object:

oclass	PCURVE, CURVE or SURFACE
mtype	PCURVE/CURVE LINE, CIRCLE, ELLIPSE, PARABOLA, HYPERBOLA, TRIMMED, BEZIER, BSPLINE, OFFSET SURFACE PLANE, SPHERICAL, CYLINDRICAL, REVOLUTION, TORIODAL, TRIMMED, BEZIER, BSPLINE, OFFSET, CONICAL, EXTRUSION
rGeom	is the reference geometry object (if none this is returned as NULL)
pinfo	is a returned pointer to the block of integer information. Filled for either BEZIER or BSPLINE, and when nonNULL is *freeable.*
prv	is the returned pointer to a block of double precision reals. The content and length depends on the oclass/mtype (*freeable*).

Note: both pinfo and prv can be NULL and the data will not be returned

API – Geometry

- **makeGeometry**

```
icode = EG_makeGeometry(ego contxt, int oclass, int mtype, ego rGeom,
                        int *pinfo, double *prv, ego *geom)
icode = IG_makeGeometry(I*8 contxt, I*4 oclass, I*4 mtype, I*8 rGeom,
                        CPTR pinfo, CPTR    prv, I*8 geom)
```

Creates a geometric object:

contxt	the **CONTEXT** object
oclass	PCURVE, CURVE or SURFACE
mtype	PCURVE/CURVE LINE, CIRCLE, ELLIPSE, PARABOLA, HYPERBOLA, TRIMMED, BEZIER, BSPLINE, OFFSET SURFACE PLANE, SPHERICAL, CYLINDRICAL, REVOLUTION, TORIODAL, TRIMMED, BEZIER, BSPLINE, OFFSET, CONICAL, EXTRUSION
rGeom	is the reference geometry object (if none use NULL)
pinfo	is a pointer to the block of integer information. Required for either BEZIER or BSPLINE.
prv	is the pointer to a block of double precision reals. The content and length depends on the oclass/mtype.
geom	is the resultant new geometry object

API – Geometry

- **getRange**

```
icode = EG_getRange(ego object, double *range, int *periodic)
icode = IG_getRange(I*8 object, R*8     range, I*4  periodic)
```

Returns the valid range of the object:

object	may be one of PCURVE, CURVE, SURFACE, EDGE or FACE
range	for PCURVE, CURVE or EDGE returns 2 values:
	t-start and *t*-end
	for SURFACE or FACE returns 4 values:
	u-min, *u*-max, *v*-min and *v*-max
periodic:	0 for non-periodic, 1 for periodic in *t* or *u*
	2 for periodic in *v* (or-able)

- **arcLength**

```
icode = EG_arcLength(ego obj, double t1, double t2, double *alen)
icode = IG_arcLength(I*8 obj, R*8    t1, R*8    t2, R*8     alen)
```

Returns the arc-length of an object.

obj	may be one of PCURVE, CURVE or EDGE
t1	starting *t*
t2	terminating *t* for calculation
alen	arc-length (returned)

API – Geometry

- **evaluate**

```
icode = EG_evaluate(ego object, double *parms, double *eval)
icode = IG_evaluate(I*8 object, R*8     parms, R*8     eval)
```

Returns the result of evaluating on the object:

object one of PCURVE, CURVE, SURFACE, NODE, EDGE or FACE
parms parameter(s) used to evaluate the object (may be NULL for NODE):
for PCURVE, CURVE or EDGE the one value is *t*
for SURFACE or FACE the 2 values are *u* then *v*
eval the returned position, 1st and 2nd derivatives (length):

	Node *(3)*	PCurve *(6)*	Edge -or- Curve *(9)*	Face -or- Surface *(18)*
Position	*[x,y,z]*	*[u,v]*	*[x,y,z]*	*[x,y,z]*
1st Derivative	--	*[du,dv]*	*[dx,dy,dz]*	*[dx_u,dy_u,dz_u]* *[dx_v,dy_v,dz_v]*
2nd Derivative	--	*[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}]* *[$dx_v{}^2$,$dy_v{}^2$,$dz_v{}^2$]*

Note: You cannot evaluate on a Degenerate EDGE

API – Geometry

- **invEvaluate**

```
icode = EG_invEvaluate(ego object,    double *pos,
                       double *parms, double *result)
icode = IG_invEvaluate(I*8 object,    R*8     pos,
                       R*8     parms, R*8     result)
```

Returns the result of inverse evaluation on the object. For topology the result is limited to inside the EDGE/FACE valid bounds.

object	may be one of PCURVE, CURVE, SURFACE, EDGE or FACE
pos	is *[u,v]* for a PCURVE and *[x,y,z]* for all others
parms	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 (2) and *[x,y,z]* for all others (3)

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

API – Geometry

- **curvature**

```
icode = EG_curvature(ego object, double *parms, double *crva)
icode = IG_curvature(I*8 object, R*8     parms, R*8     crva)
```

Returns the curvature and principle directions/tangents:

object may be one of PCURVE, CURVE, SURFACE, EDGE or FACE
parms parameter(s) used to evaluate on the object:
for PCURVE, CURVE or EDGE the one value is *t*
for SURFACE or FACE the 2 values are *u* then *v*
crva the returned curvature information (length):

PCurve *(3)*	Edge -or- Curve *(4)*	Face -or- Surface *(8)*
curvature	curvature	curvature1
[dir.x, dir.y]	*[dir.x ,dir.y, dir.z]*	*[dir1.x, dir1.y, dir1.z]*
		curvature2
		[dir2.x, dir2.y, dir2.z]

API – Geometry

- **approximate**

```
icode = EG_approximate(ego context, int mDeg, double tol,
                       int *sizes, double *xyz, ego *geo)
icode = IG_approximate(I*8 context, I*4 mDeg, R*8    tol,
                       I*4  sizes, R*8     xyz, I*8  geo)
```

Computes and returns the resultant geometry object created by approximating the data by a BSpline (OCC or EGADS method).

context	the **CONTEXT** object used to place the result
mDeg	the maximum degree used by OCC [3-8], or cubic by EGADS [0-2] 0 – fixes the bounds and uses natural end conditions 1 – fixes the bounds and maintains the slope input at the bounds 2 – fixes the bounds & quadratically maintains the slope at 2nd order
tol	is the tolerance to use for the BSpline approximation procedure, zero for a **SURFACE** fit (OCC).
sizes	a vector of 2 integers that specifies the size and dimensionality of the data. If the second is zero, then a CURVE is fit and the first integer is the length of the number of *[x,y,z]* triads. If the second integer is nonzero then the input data reflects a 2D map.
xyz	the data to fit (3 times the number of points in length)
geo	the returned approximated (or fit) BSpline resultant object

API – Geometry

- **fitTriangles**

```
icode = EG_fitTriangles(ego context, int  len,   double *pxyz,
                        int ntri,    int *ptris, int *ptric,
                        double tol,  ego *geo)
icode = IG_fitTriangles(I*8 context, I*4  len,   R*8     pxyz,
                        I*4 ntri,    I*4  ptris, int  ptric,
                        R*8    tol,  I*8  geo)
```

Computes and returns the resultant geometry object created by approximating the triangulation by a BSpline surface.

context	the **CONTEXT** object used to place the result
len	the number of vertices in the triangulation
pxyz	the coordinates to fit (3 times len in length)
ntri	the number of triangles
ptris	the pointer to triangle indices (1 bias) (3 times ntri in length)
ptric	the pointer to neighbor triangle indices (1 bias) -- 0 or (-) at bounds NULL -- will compute (3 times ntri in length, if not NULL)
tol	is the tolerance to use for the BSpline approximation procedure
geo	the returned approximated BSpline resultant object

API – Geometry

- **otherCurve**

```
icode = EG_otherCurve(ego object, ego iCrv, double tol,
                      ego *oCrv)
icode = IG_otherCurve(I*8 object, I*8 iCrv, R*8    tol,
                      I*8  oCrv)
```

Computes and returns the *other* curve that matches the input curve. If the input curve is a PCURVE, the output is a 3D CURVE (and *vice versa*).

object	the **SURFACE** or **FACE** object used for the conversion
iCrv	the input **PCURVE** or **CURVE/EDGE** object
tol	is the tolerance to use when fitting the output curve
oCrv	the returned approximated resultant curve object

- **isSame**

```
icode = EG_isSame(ego obj1, ego obj2)
icode = IG_isSame(I*8 obj1, I*8 obj2)
```

Compares two objects for geometric equivalence.

obj1	an object of type **NODE, CURVE**, **EDGE**, **SURFACE** or **FACE**
obj2	an object of the same dimensionality

API – Geometry

- **isoCline**

```
icode = EG_isoCline(ego surface, int iUV, double value,
                    ego *oCrv)
icode = IG_isoCline(I*8 surface, I*4 iUV, R*8    value,
                    I*8  oCrv)
```

Computes from the input Surface and returns the isocline curve.

surface	the **SURFACE** object used for the source
iUV	the type of isocline: **UISO** (0) constant U or **VISO** (1) constant V
value	the value used for the isocline
oCrv	the returned resultant curve object

- **convertToBSpline**

```
icode = EG_convertToBSpline(ego geom, ego *bspline)
icode = IG_convertToBSpline(I*8 geom, I*8  bspline)
```

Computes and returns the BSpline representation of the input geometric object.

geom	can be a **PCURVE**, **CURVE**, **EDGE**, **SURFACE** or **FACE**
bspline	the returned approximated resultant **BSPLINE** object

API – Topology

- **getTopology**

```
icode = EG_getTopology(ego object, ego    *ref,  int *oclass,
                       int *mtype, double *data, int *nchild,
                       ego **pchldrn, int **psens)
icode = IG_getTopology(I*8 object, I*8     ref,  I*4  oclass,
                       I*4 mtype,  R*8     data, I*4  nchild,
                       CPTR  pchldrn, CPTR  psens)
```

Returns information about the topological object:

ref	is the reference geometry object (if none this is returned as NULL)
oclass	is NODE, EDGE, LOOP, FACE, SHELL, BODY or MODEL
mtype	for EDGE is TWONODE, ONENODE or DEGENERATE
	for LOOP is OPEN or CLOSED
	for FACE is either SFORWARD or SREVERSE
	for SHELL is OPEN or CLOSED
	BODY is either WIREBODY, FACEBODY, SHEETBODY or SOLIDBODY
data	will retrieve at most 4 doubles:
	for NODE this contains the *[x,y,z]* location
	EDGE is the *t*-min and *t*-max (the parametric bounds)
	FACE returns the *[u,v]* box (the limits first for *u* then for *v*)
nchild	number of children (lesser) topological objects
pchldrn	is a returned pointer to the block of children objects.
	FORTRAN only note: this pointer is *freeable.*
psens	is the returned pointer to a block of integer senses for the children.

API – Topology

- **makeTopology**

```
icode = EG_makeTopology(ego context, ego ref,  int oclass,
                        int mtype, double *data, int nchild,
                        ego *chldrn, int *senses, ego *topo)
icode = IG_makeTopology(I*8 context, I*8 ref,  I*4 oclass,
                        I*4 mtype, R*8     data, I*4 nchild,
                        I*8  chldrn, I*4  senses, I*8  topo)
```

Creates and returns a topological object:

context	the CONTEXT object used to place the result
ref	reference geometry object required for EDGEs and FACEs (optional for LOOP)
oclass	is either NODE, EDGE, LOOP, FACE, SHELL, BODY or MODEL
mtype	for EDGE is TWONODE, ONENODE or DEGENERATE
	for LOOP is OPEN or CLOSED
	for FACE is either SFORWARD or SREVERSE
	for SHELL is OPEN or CLOSED
	BODY is either WIREBODY, FACEBODY, SHEETBODY or SOLIDBODY
data	may be NULL except for:
	NODE which contains the *[x,y,z]* location
	EDGE is the *t*-min and *t*-max (the parametric bounds)
nchild	number of children (lesser) topological objects
chldrn	a vector of children objects (nchild in length)
	if LOOP and has reference SURFACE, then 2*nchild in length (PCURVES follow)
senses	a vector of integer senses for the children (required for FACES & LOOPs only)
topo	the resultant returned topological object

API – Topology

- **makeFace**

```
icode = EG_makeFace(ego object, int mtype, double *data,
                    ego *face)
icode = IG_makeFace(I*8 object, I*4 mtype, R*8     data,
                    I*8  face)
```

Creates a simple FACE from a LOOP or a SURFACE. Also can be used to hollow a single LOOPed existing FACE. This function creates any required NODEs, EDGEs and LOOPs.

object	either a LOOP (for a planar *cap*), a SURFACE with *[u,v]* bounds, or a FACE to be hollowed out
mtype	is either SFORWARD or SREVERSE for LOOPs you may want to look at the orientation using **getArea**, ignored when the input object is a FACE
data	may be NULL for LOOPs, but must be the limits for a SURFACE (4 values), the hollow/offset distance and fillet radius (zero is for no fillets) for a FACE input object (2 values)
face	the resultant returned topological FACE object (a return of **EGADS_OUTSIDE** is the indication that offset distance was too large to produce any cutouts, and this result is the input object)

API – Topology

- **makeLoop**

```
icode = EG_makeLoop(int nEdge,    ego *edges, ego geom,
                    double toler, ego *loop)
icode = IG_makeLoop(I*4 nEdge,    I*8  edges, I*8 geom,
                    R*8    toler, I*8  loop)
```

Creates a LOOP from a list of EDGE Objects, where the EDGEs do not have to be topologically connected. The tolerance is used to build the NODEs for the LOOP. The orientation is set by the first non-NULL entry in the list, which is taken in the positive sense. This is designed to be executed until all list entries are exhausted.

nEdge	the number of EDGE Objects in the list (>= 1)
edges	list of EDGEs, of which some may be NULL (nEdge in length) Note: list entries are NULLified when included in LOOPs
geom	SURFACE Object for non-planar LOOPs to be used to bound FACEs (can be NULL)
toler	tolerance used for the operation (0.0 - use EDGE tolerances)
loop	the resultant LOOP Object
icode	the number of nonNULL entries in edges when returned or error code

API – Topology

- **makeSolidBody**

```
icode = EG_makeSolidBody(ego context,  int stype,
                         double *data, ego *body)
icode = IG_makeSolidBody(I*8 context,  I*4 stype,
                         R*8     data, I*8  body)
```

Creates a simple SOLIDBODY. Can be either a box, cylinder, sphere, cone, or torus.

context	the CONTEXT object used to place the result
stype	1-box, 2-sphere, 3-cone, 4-cylinder, or 5-torus
data	depends on stype:
	box (6): *[x,y,z]* then *[dx,dy,dz]* for size of box
	sphere (4): *[x,y,z]* of center then radius
	cone (7): apex *[x,y,z]*, base center *[x,y,z]*, then radius
	cylinder (7): 2 axis points and the radius
	torus (8): *[x,y,z]* of center, direction of rotation, then major radius and minor radius
body	the resultant returned topological BODY object

API – Topology

- **getBodyTopos**

```
icode = EG_getBodyTopos(ego body,   ego ref, int oclass,
                        int *ntopo, ego **ptopos)
icode = IG_getBodyTopos(I*8 body,   I*8 ref, I*4 oclass,
                        I*4  ntopo, CPTR  ptopos)
```

Returns topologically connected objects:

body	body container object
ref	reference topological object or NULL. Sets the context for the returned objects (i.e. all objects of a 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*
ntopo	the returned number of requested topological objects
ptopos	is a returned pointer to the block of objects (*freeable*) can be NULL and will not be filled

API – Topology

- **indexBodyTopo**

```
index = EG_indexBodyTopo(ego body, ego obj)
index = IG_indexBodyTopo(I*8 body, I*8 obj)
```

Returns the index (bias 1) of the topological object in the Body:

body	body container object
obj	is the topological object in the Body
index	the index or an error indicator (negative)

- **objectBodyTopo**

```
icode = EG_objectBodyTopo(ego body, int oclass, int index,
                          ego *obj)
icode = IG_objectBodyTopo(I*8 body, I*4 oclass, I*4 index,
                          I*8  obj)
```

Returns the topological object (based on index) in the Body:

body	body container object
oclass	is NODE, EDGE, LOOP, FACE or SHELL
index	is the index (bias 1) of the entity requested
obj	is the returned topological object in the Body

API – Topology

- **getArea**

```
icode = EG_getArea(ego object, double *data, double *area)
icode = IG_getArea(I*8 object, R*8     data, R*8     area)
```

Computes the surface area from a LOOP, a SURFACE or a FACE. When a LOOP is used a planar surface is fit and the resultant area can be negative if the orientation of the fit is opposite of the LOOP.

object	either a LOOP (for a planar *cap*), a SURFACE with *[u,v]* bounds or a FACE
data	may be NULL except must contain the limits for a SURFACE
area	the resultant surface area returned

- **getBoundingBox**

```
icode = EG_getBoundingBox(ego object, double *box)
icode = IG_getBoundingBox(I*8 object, R*8     box)
```

Computes the Cartesian bounding box around the object:

object	any topological object
box	6 doubles reflecting the *[x,y,z]* min and *[x,y,z]* max

API – Topology

- **getMassProperties**

```
icode = EG_getMassProperties(ego topo, double *data)
icode = IG_getMassProperties(I*8 topo, R*8     data)
```

Computes and returns the physical and inertial properties of a topological object.

topo	the object, can be EDGE, LOOP, FACE, SHELL or BODY
data	the data returned (must be declared to at least 14 doubles): volume, surface area (length for EDGE, LOOP or WIREBODY) center of gravity (3) inertia matrix at CoG (9)

- **isEquivalent**

```
icode = EG_isEquivalent(ego topo1, ego topo2)
icode = IG_isEquivalent(I*8 topo1, I*8 topo2)
```

Compares two topological objects for equivalence.

topo1	a topological object
topo2	a topological object of the same class

API – Topology

- **inTopology**

```
icode = EG_inTopology(ego topo, double *xyz)
icode = IG_inTopology(I*8 topo, R*8     xyz)
```

Computes whether the point is on or contained within the object. Works with EDGEs and FACEs by projection. SHELLs must be CLOSED.

topo	the object, can be EDGE, FACE, SHELL or SOLIDBODY
xyz	the coordinate location to check
icode	the result or error code

- **inFace**

```
icode = EG_inFace(ego face, double *uv)
icode = IG_inFace(I*8 face, R*8     uv)
```

Computes the result of the *[u,v]* location in the valid part of the FACE.

face	the FACE object
uv	the parametric location to check
icode	the result or error code

API – Topology

- **getEdgeUV**

```
icode = EG_getEdgeUV(ego face, ego edge, int sense,
                     double t, double *uv)
icode = IG_getEdgeUV(I*8 face, I*8 edge, I*4 sense,
                     R*8    t, R*8     uv)
```

Computes on the EDGE/PCURVE to get the appropriate *[u,v]* on the FACE.

face	the FACE object
edge	the EDGE object
sense	can be 0, but must be specified (+/-1) if the EDGE is found the FACE twice that denotes the position in the LOOP to use. EGADS_TOPOERR is returned when sense==0 and an EDGE is found twice.
t	the parametric value to use for the evaluation
uv	the resulting *[u,v]* evaluated at *t*.

- **getBody**

```
icode = EG_getBody(ego object, ego *body)
icode = IG_getBody(I*8 object, I*8  body)
```

Returns the Body containing the Object.

object	the input object
body	the returned Body object (if in a Body, otherwise NULL)

API – Topology

- **sewFaces**

```
icode = EG_sewFaces(int nObject, ego *objects, double tol,
                    int flag, ego *model)
icode = IG_sewFaces(I*4 nObject, I*8  objects, R*8    tol,
                    int flag, I*8  model)
```

Creates a MODEL from a collection of Objects. The Objects can be either BODYs (not WIREBODY), SHELLs and/or FACEs. After the sewing operation, any unconnected Objects are returned as BODYs.

nObject	the number of Objects in the list
objects	list of Objects to sew together (nObject in length)
toler	tolerance used for the operation (0.0 - use Face tolerances)
flag	0 - manifold, 1 - allow non-manifold results
model	the resultant MODEL object

API – Topology / Tolerance

- **getTolerance**

```
icode = EG_getTolerance(ego object, double *tol)
icode = IG_getTolerance(I*8 object, R*8     tol)
```

Returns the internal tolerance defined for the object.

object	topological object (all except MODEL)
tol	the tolerance used to define closure

- **tolerance**

```
icode = EG_tolerance(ego object, double *tol)
icode = IG_tolerance(I*8 object, R*8     tol)
```

Returns the maximum tolerance defined for the object's hierarchy.

object	topological object (all except MODEL)
tol	the tolerance used to define closure

- **setTolerance**

```
icode = EG_setTolerance(ego object, double tol)
icode = IG_setTolerance(I*8 object, R*8    tol)
```

Sets the internal tolerance defined for the object. Useful for SBOs.

object	topological object (all except MODEL)
tol	the tolerance used to define closure

API – Topology

- **replaceFaces**

```
icode = EG_replaceFaces(ego body, int nFace, ego *faces,
                        ego *result)
icode = IG_replaceFaces(I*8 body, I*4 nFace, I*8  faces,
                        I*8  result)
```

Creates a new SHEETBODY or SOLIDBODY from an input SHEETBODY or SOLIDBODY and a list of FACEs to modify. The FACEs are input in pairs where the first must be an Object in the BODY and the second either a new FACE or NULL. The NULL replacement flags removal of the FACE in the BODY.

body	body container object Note: SOLIDBODYs must have a single (outer) SHELL
nFace	the number of FACE pairs in the list
faces	list of FACE pairs, where the first must be a FACE in the BODY and second is either the FACE to use as a replacement or a NULL which indicates that the FACE is to be removed from the BODY 2*nFace in length
result	the resultant BODY object, either a SHEETBODY or a SOLIDBODY (where the input was a SOLIDBODY and all FACEs are replaced in a way that the LOOPs match up)

API – Topology

- **matchBodyEdges**

```
icode = EG_matchBodyEdges(ego body1, ego body2, double tol,
                          int *nMatch, int **matches)
icode = IG_matchBodyEdges(I*8 body1, I*8 body2, R*8    tol,
                          I*4  nMatch, CPTR  matches)
```

Examines the EDGEs in one BODY against all of the EDGEs in another. If the number of NODEs, the NODE locations, the EDGE bounding boxes and the EDGE arc lengths match it is assumed that the EDGEs match. A list of pairs of indices are returned.

body1	first body container object
body2	second body container object
tol	the tolerance used (can be zero to use entity tolerances)
nMatch	the number of matched EDGE pairs in the list
matches	pointer to a list of EDGE pairs, returned as NULL if nMatch is zero, otherwise it is a pointer to 2*nMatch integers, where each pair is the matching indices in the respective bodies (freeable)

API – Topology

- **matchBodyFaces**

```
icode = EG_matchBodyFaces(ego body1, ego body2, double tol,
                          int *nMatch, int **matches)
icode = IG_matchBodyFaces(I*8 body1, I*8 body2, R*8    tol,
                          I*4  nMatch, CPTR  matches)
```

Examines the FACEs in one BODY against all of the FACEs in another. If the number of LOOPs, number of NODEs, the NODE locations, the number of EDGEs and the EDGE bounding boxes as well as the EDGE arc lengths match it is assumed that the FACEs match. A list of pairs of indices are returned.

body1	first body container object
body2	second body container object
tol	the tolerance used (can be zero to use entity tolerances)
nMatch	the number of matched FACE pairs in the list
matches	pointer to a list of FACE pairs, returned as NULL if nMatch is zero, otherwise it is a pointer to 2*nMatch integers, where each pair is the matching indices in the respective bodies (freeable)

Note: This is useful for the situation where there are *glancing* FACEs and a UNION operation fails (or would fail). Simply find the matching FACEs and do not include them in a call to **`EG_sewFaces`**.

API – Topology / Tessellation

- **mapBody**

```
icode = EG_mapBody(ego src, ego dst, char *fAttr,
                   ego *mapped)
icode = IG_mapBody(I*8 src, I*8 dst, C**   fAttr,
                   I*8  mapped)
```

Checks for topological equivalence between the the BODY *src* and the BODY *dst*. If necessary, produces a mapping (indices in *src* which map to *dst*) and places these as attributes on the resultant BODY *mapped* (named *.nMap*, *.eMap* and *.fMap*). Also may modify BSplines associated with FACEs.

src	source body object (not WIREBODY)
dst	destination body object
fAttr	the FACE attribute used to map FACEs
mapped	the mapped resultant BODY object copied from *dst* If NULL and *icode* == EGADS_SUCCESS, *dst* is equivalent and can be used directly in **EG_mapTessBody**

Note: It is the responsibility of the caller to have uniquely attributed all FACEs in both *src* and *dst* to aid in the mapping for all but FACEBODYs.

API – Tessellation

- **makeTessGeom**

```
icode = EG_makeTessGeom(ego geom, double *limits, int *sizes,
                        ego *tess)
icode = IG_makeTessGeom(I*8 geom, R*8     limits, I*4  sizes,
                        I*8  tess)
```

Creates a discretization object from a geometry-based Object.

geom	the input object, may be a CURVE or SURFACE
limits	the bounds of the tessellation (like range)
sizes	a set of 2 integers that specifies the size and dimensionality of the data. The second is assumed zero for a CURVE and in this case the first integer is the length of the number of evenly spaced (in *t*) points created. The second integer must be nonzero for SURFACEs and this then specifies the density of the *[u,v]* map of coordinates produced (again evenly spaced in the parametric space). If a value of sizes is negative, then the fill is reversed for that coordinate.
tess	the resultant TESSELLATION object

API – Tessellation

- **getTessGeom**

```
icode = EG_getTessGeom(ego tess, int *sizes, double **pxyz)
icode = IG_getTessGeom(I*8 tess, I*4  sizes, CPTR     pxyz)
```

Retrieves the data associated with the discretization of a geometry-based Object.

tess	the TESSELLATION object
sizes	a returned set of 2 integers that specifies the size and dimensionality of the data. If the second is zero, then it is from a CURVE and the first integer is the length of the number of *[x,y,z]* triads. If the second integer is nonzero then the input data reflects a 2D map of coordinates.
pxyz	the returned pointer to the suite of coordinate data.

API – Tessellation

- **makeTessBody**

```
icode = EG_makeTessBody(ego body, double *parms, ego *tess)
icode = IG_makeTessBody(I*8 body, R*8     parms, I*8  tess)
```

Creates a discretization object from a Topological BODY Object.

body	the input object, may be any Body type.
parms	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.
tess	the resultant TESSELLATION object where each EDGE in the BODY is discretized and each FACE is triangulated.

Notes:

1. the attribute ".tParams" on the BODY, FACEs or EDGEs overrides **parms** locally (the minimum is used). This attribute must be REAL, have 3 values (as described above).
2. the REAL attribute ".tPos" directly sets the *t*s for interior EDGE positions.
3. the REAL attribute ".rPos" sets the relative spacing (in *arc-length*) for interior EDGE positions.
4. the INTEGER attribute ".nPos" sets the relative spacing (evenly in *arc-length*) for interior EDGE positions.
5. an INTEGER attribute ".tPos" or ".rPos" of length 1 and containing a zero -- no interior points.

API – Tessellation

- **remakeTess**

```
icode = EG_remakeTess(ego tess, int nobj, ego *facedg, double *parms)
icode = IG_remakeTess(I*8 tess, I*4 nobj, I*8  facedg, R*8     parms)
```

Redoes the discretization for specified objects from within a BODY TESSELLATION.

tess	the TESSELLATION object to modify.
nobj	number of objects in the face/edge list.
facedg	list of FACE and/or EDGE objects from within the BODY used to create the TESSELLATION object. First all specified Edges are rediscretized. Then any listed Face and the Faces touched by the retessellated Edges are retriangulated. Note that Quad Patches associated with Faces whose Edges were redone will be removed.
parms	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.

API – Tessellation

- **finishTess**

```
icode = EG_finishTess(ego tess, double *parms)
icode = IG_finishTess(I*8 tess, R*8     parms)
```

Completes the discretization for specified objects for the input TESSELLATION object.

tess	the open TESSELLATION object to finish.
parms	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.

Note: an open TESSELLATION object is created by **`EG_initTessBody`** and can be partially filled via **`EG_setTessEdge`** and/or **`EG_setTessFace`** before this function is invoked.

API – Tessellation

- **mapTessBody**

```
icode = EG_mapTessBody(ego tess, ego body, ego *mapTess)
icode = IG_mapTessBody(I*8 tess, I*8 body, I*8  mapTess)
```

Maps the input discretization object to another BODY Object. The topologies of the BODY that created the input tessellation must match the topology of the body argument (the use of **`EG_mapBody`** can be used to assist).

tess	the input BODY TESSELLATION object
body	the BODY object (with a matching Topology) used to map the tessellation.
mapTess	the resultant TESSELLATION object. The triangulation is simply copied but the *uv* and *xyz* positions reflect the input body (above).

Note: Invoking **`EG_moveEdgeVert`**, **`EG_deleteEdgeVert`** and/or **`EG_insertEdgeVerts`** in the source tessellation before calling this routine invalidates the ability of **`EG_mapTessBody`** to perform its function.

API – Tessellation

- **locateTessBody**

```
icode = EG_locateTessBody(ego tess, int npts, int *ifaces,
                          double *uvs, int *itris, double *results)
icode = IG_locateTessBody(I*8 tess, I*4 npts, I*4  ifaces,
                          R*8     uvs, I*4  itris, R*8     results)
```

Provides the triangle and the vertex weights for each of the input requests or the evaluated positions in a mapped tessellation

tess	the input BODY TESSELLATION object
npts	the number of input requests
ifaces	the face indices for each request – minus index refers to the use of a mapped Face index from **`EG_mapBody`** and **`EG_mapTessBody`** (npts in length)
uvs	the UV positions in the face for each request (2*npts in length)
itris	the resultant 1-bias triangle index (npts in length) if input as NULL then this function will perform mapped evaluations
results	the vertex weights in the triangle that refer to the requested position (any negative weight indicates that the point was extrapolated) -or- the evaluated position based on the input uvs (when itris is NULL) (3*npts in length)

API – Tessellation

- **getTessEdge**

```
icode = EG_getTessEdge(ego tess, int eIndex, int *len,
                       double **pxyz, double **pt)
icode = IG_getTessEdge(I*8 tess, I*4 eIndex, I*4  len,
                       CPTR     pxyz, CPRT     pt)
```

Retrieves the data associated with the discretization of an EDGE from a Body-based Tessellation Object.

tess	the 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 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
pxyz	the returned pointer to the set of coordinate data.
pt	the returned pointer to the parameter values associated with each vertex.

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

API – Tessellation

- **getTessFace**

```
icode = EG_getTessFace(ego tess, int fIndex, int *len, double **pxyz,
                       double **puv, int **ptype, int **pindex,
                       int *ntri, int **ptris, int **ptric)
icode = IG_getTessFace(I*8 tess, I*4 fIndex, I*4  len, CPTR     pxyz,
                       CPRT     puv, CPRT  ptype, CPRT  pindex,
                       I*4  ntri, CPTR  ptris, CPRT  ptric)
```

Retrieves the data associated with the discretization of a FACE from a Body-based Tessellation Object.

tess	the TESSELLATION object
fIndex	the FACE index (1 bias). The FACE Objects and number of FACEs can be retrieved via **`EG_getBodyTopos`** and/or **`EG_indexBodyTopo`**. A minus refers to the use of a mapped (+) FACE index (if it exists).
len	the returned number of vertices in the triangulation
pxyz	the returned pointer to the set of coordinate data for each vertex
puv	returned pointer to the parameter values associated with each vertex
ptype	returned pointer to the vertex type (-1 - internal, 0 - NODE, >0 EDGE)
pindex	returned pointer to vertex index (-1 internal)
ntri	returned number of triangles
ptris	returned pointer to triangle indices, 3 per triangle (1 bias) orientation consistent with FACE's **`mtype`**
ptric	returned pointer to neighbor information, 3 per triangle

API – Tessellation

- **getTessLoops**

```
icode = EG_getTessLoops(ego tess, int fIndex, int *nloop,
                        int **lIndex)
icode = IG_getTessLoops(I*8 tess, I*4 fIndex, I*4  nloop,
                        CPTR  lIndex)
```

Retrieves the data for the LOOPs associated with the discretization of a FACE from a Body-based Tessellation Object.

tess	the TESSELLATION object
fIndex	the FACE index (1 bias). The FACE Objects and number of FACEs can be retrieved via **`EG_getBodyTopos`** and/or **`EG_indexBodyTopo`**.
nloop	the returned number of LOOPs in the FACE triangulation
lIndex	the returned pointer to a vector of the last index (bias 1) for each LOOP (nloop in length). Notes: (1) all boundary vertices are listed first for any FACE tessellation, (2) outer LOOP is ordered in the counter-clockwise direction, and (3) inner LOOP(s) are ordered in the clockwise direction.

API – Tessellation

- **getTessQuads**

```
icode = EG_getTessQuads(ego tess, int *len, int **pindices)
icode = IG_getTessQuads(I*8 tess, I*4  len, CPTR  pindices)
```

Returns a list of FACE indices found in the Body-based Tessellation Object that has been successfully *Quadded*.

tess	the TESSELLATION object
len	the returned number of FACEs with Quad patches
pindices	the returned pointer the FACE indices (1 bias). The FACE Objects themselves can be retrieved via **`getBodyTopos`**. This pointer is *freeable*.

API – Tessellation

- **makeQuads**

```
icode = EG_makeQuads(ego tess, double *parms, int fIndex)
icode = IG_makeQuads(I*8 tess, R*8     parms, I*4 fIndex)
```

Creates Quadrilateral Patches for the indicated FACE and updates the Body-based Tessellation Object.

tess	the TESSELLATION object
parms	a set of 3 parameters that drive the Quadrilateral patching for the FACE. Any may be set to zero to indicate the use of the default value:
	parms[0] EDGE matching tolerance expressed as the deviation from an aligned dot product [default: *0.05*]
	parms[1] Maximum quad *side* ratio point count to allow [default: *3.0*]
	parms[2] Number of smoothing loops [default: *0.0*]
fIndex	the FACE index (1 bias)

API – Tessellation

- **getQuads**

```
icode = EG_getQuads(ego tess, int fIndex, int *len,
                    double **pxyz, double **puv, int **ptype,
                    int **pindex, int *npatch)
icode = IG_getQuads(I*8 tess, I*4 fIndex, I*4  len,
                    CPTR     pxyz, CPRT     puv, CPRT  ptype,
                    CPRT  pindex, I*4  npatch)
```

Retrieves the data associated with the Quad-patching of a FACE from a Body-based Tessellation Object.

tess	the TESSELLATION object
fIndex	the FACE index (1 bias). The FACE Objects and number of FACEs can be retrieved via **`getBodyTopos`**.
len	the returned number of vertices in the patching
pxyz	the returned pointer to the set of coordinate data for each vertex
puv	returned pointer to the parameter values associated with each vertex
ptype	returned pointer to the vertex type (-1 - internal, 0 - NODE, >0 EDGE)
pindex	returned pointer to vertex index (-1 internal)
npatch	returned number of patches

API – Tessellation

- **getPatch**

```
icode = EG_getPatch(ego tess, int fIndex, int pIndex, int *n1,
                    int *n2, int **pvindex, int **pbounds)
icode = IG_getPatch(I*8 tess, I*4 fIndex, I*4 pIndex, I*4  n1,
                    I*4  n2, CPRT  pvindex, CPRT  pbounds)
```

Retrieves the data associated with the Patch of a FACE from a Body-based Tessellation Object.

tess	the TESSELLATION object
fIndex	the FACE index (1 bias). The FACE Objects and number of FACEs can be retrieved via **`getBodyTopos`**.
pIndex	the patch index (**1-npatch** from **`EG_getQuads`**)
n1	the returned patch size in the first direction (indexed by *i*)
n2	the returned patch size in the second direction (indexed by *j*)
pvindex	the returned pointer to *n1*n2* indices that define the patch
pbounds	returned pointer to the neighbor bounding information for the patch (*2*(n1-1)+2*(n2-1)* in length). The first represents the segments at the base (*j* at base and increasing in *i*), the next is at the right (with *i* at max and *j* increasing). The third is the top (with *j* at max and *i* decreasing) and finally the left (*i* at min and *j* decreasing).

API – Tessellation

- **moveEdgeVert**

```
icode = EG_moveEdgeVert(ego tess, int eIndex, int vIndex, double t)
icode = IG_moveEdgeVert(I*8 tess, I*4 eIndex, I*4 vIndex, R*8    t)
```

Moves the position of an EDGE vertex in a Body-based Tessellation Object. Will invalidate the *Quad* patches on any FACEs touching the EDGE.

tess	the TESSELLATION object (not on WIREBODIES)
eIndex	the EDGE index (1 bias).
vIndex	the Vertex index in the EDGE (2 - nVert-1)
t	the new parameter value on the EDGE for the point

- **deleteEdgeVert**

```
icode = EG_deleteEdgeVert(ego tess, int eIndex, int vIndex, int dir)
icode = IG_deleteEdgeVert(I*8 tess, I*4 eIndex, I*4 vIndex, I*4 dir)
```

Deletes an EDGE vertex from a Body-based Tessellation Object. Will invalidate the *Quad* patches on any FACEs touching the EDGE.

tess	the TESSELLATION object (not on WIREBODIES)
eIndex	the EDGE index (1 bias).
vIndex	the Vertex index in the EDGE to delete (2 - nVert-1)
dir	the direction to collapse any triangles (either -1 or 1)

API – Tessellation

- **insertEdgeVerts**

```
icode = EG_insertEdgeVerts(ego tess, int eIndex, int vIndex,
                           int len, double *ts)
icode = IG_insertEdgeVerts(I*8 tess, I*4 eIndex, I*4 vIndex,
                           I*4 len, R*8     ts)
```

Inserts vertices into the EDGE discretization of a Body Tessellation Object. This will invalidate the *Quad* patches on any FACEs touching the EDGE.

tess	the TESSELLATION object (not on WIREBODIES)
eIndex	the EDGE index (1 bias).
vIndex	the Vertex index in the EDGE to insert the points after (1 - nVert-1)
len	the number of points to insert
ts	the ***t*** values for the new points. Must be monotonically increasing and be greater than the ***t*** of vIndex and less than the ***t*** of vIndex+1.

- **openTessBody**

```
icode = EG_openTessBody(ego tess)
icode = IG_openTessBody(I*8 tess)
```

Opens an existing Tessellation Object for replacing EDGE/FACE discretizations.

tess	the TESSELLATION object to open for edits.

API – Tessellation

- **initTessBody**

```
icode = EG_initTessBody(ego body, ego *tess)
icode = IG_initTessBody(I*8 body, I*8  tess)
```

Creates an empty (open) discretization object for a Topological BODY Object.

body the input object, may be any Body type.

tess resultant empty TESSELLATION object where each EDGE in the BODY must be filled via a call to **`EG_setTessEdge`** and each FACE must be filled with invocations of **`EG_setTessFace`**. The TESSSELLATION object is considered open until all EDGEs have been set (for a WIREBODY) or all FACEs have been set (for other Body types).

- **statusTessBody**

```
icode = EG_statusTessBody(ego tess, ego *body, int *stat, int *npts)
icode = IG_statusTessBody(I*8 tess, I*8  body, I*4  stat, i*4  npts)
```

Returns the status of a TESSELLATION Object.

tess the TESSELLATION Object to query.

body the returned associated BODY Object.

stat the state of the tessellation: 0 - open, 1 - OK, 2 - displaced.

npts the number of global points in the tessellation (0 -- open)

icode 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

API – Tessellation

- **setTessEdge**

```
icode = EG_setTessEdge(ego tess, int eIndex, int len,
                       double *pxyz, double *pt)
icode = IG_setTessEdge(I*8 tess, I*4 eIndex, I*4 len,
                       CPTR    pxyz, CPRT    pt)
```

Sets the data associated with the discretization of an EDGE for an open Body-based Tessellation Object.

tess	the open (not complete) 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.`** If this EDGE already has assigned data, it is overwritten.
len	the number of vertices in the EDGE discretization.
pxyz	the pointer to the set of coordinate data.
pt	the pointer to the parameter values associated with each vertex.

Notes: (1) all vertices must be specified in increasing *t*.
(2) the coordinates for the first and last vertex MUST match the appropriate NODE's coordinates.
(3) problems are reported to *Standard Out* regardless of the *OutLevel*.

API – Tessellation

- **setTessFace**

```
icode = EG_setTessFace(ego tess, int fIndex, int len, double *pxyz,
                       double *puv, int ntri, int *ptris)
icode = IG_setTessFace(I*8 tess, I*4 fIndex, I*4 len, CPTR    pxyz,
                       CPRT    puv, I*4 ntri, CPTR ptris)
```

Sets the data associated with the discretization of a FACE for an open Body-based Tessellation Object.

tess	the open (not complete) TESSELLATION object.
fIndex	the FACE index (1 bias). The FACE Objects and number of FACEs can be retrieved via **`EG_getBodyTopos`** and/or **`EG_indexBodyTopo`**.
	If this FACE already has assigned data, it is overwritten.
len	the number of vertices in the triangulation.
pxyz	the pointer to the set of coordinate data for all vertices -- 3*len in length.
puv	the pointer to the vertex parameter values -- 2*len in length.
ntri	the number of triangles.
ptris	the pointer to triangle vertex indices (1 bias) -- 3*ntri in length.

Notes:
(1) all EDGEs associated with the FACE must have been set.
(2) any vertex associated with a NODE or an EDGE must use the coordinates specified by the NODE or the EDGE discretization.
(3) this function will reorder the vertices to match EGADS internal assumptions.
(4) problems are reported to *Standard Out* regardless of the *OutLevel*.

API – Tessellation

- **localToGlobal**

```
icode = EG_localToGlobal(ego tess, int index, int local, int *global)
icode = IG_localToGlobal(I*8 tess, I*4 index, I*4 local, I*8  global)
```

Perform Local to Global index lookup. Tessellation Object must be closed.

tess	the TESSELLATION object
index	the topological index (1 bias) -- (-) Edge, (+) Face
local	the local index
global	the returned global index

- **getGlobal**

```
icode = EG_getGlobal(ego tess, int global, int *pty, int *pin, double *xyz)
icode = IG_getGlobal(I*8 tess, I*4 global, I*4  pty, I*4  pin, R*8     xyz)
```

Returns the point type and index (like from EG_getTessFace) with optional coordinates.

tess	the TESSELLATION object
global	the global index (1 bias).
pty	the point type (-) Face local index, (0) Node, (+) Edge local index
pin	the point topological index (1 bias)
xyz	the coordinates at this global index (can be NULL for no return)

API – High-Level Functions

- **solidBoolean**

```
icode = EG_solidBoolean(ego src, ego tool, int oper,
                        ego *model)
icode = IG_solidBoolean(I*8 src, I*8 tool, I*4 oper,
                        I*8  model)
```

Performs the Solid Boolean Operations (SBOs) on the source BODY Object (that has the type SOLIDBODY). The tool object types depend on the operation. This supports Intersection, Subtraction and Union.

src	the source SOLIDBODY object
tool	the tool object: either a SOLIDBODY for all operators -or- a FACE/FACEBODY for Subtraction
oper	1-Subtraction, 2-Intersection and 3-Fusion
model	the resultant MODEL object (this is because there may be multiple bodies from either the subtraction or intersection operation).

Note: This may be called with *src* being a MODEL. In this case *tool* may be a SOLIDBODY for Intersection/Subtraction or a FACE/FACEBODY for Fusion. The input MODEL may contain anything, but must not have duplicate topology.

API – High-Level Functions

- **fuseSheets**

```
icode = EG_fuseSheets(ego src, ego tool, ego *sheet)
icode = IG_fuseSheets(I*8 src, I*8 tool, I*8  sheet)
```

Fuses (unions) two SHEETBODYs resulting in a single SHEETBODY.

src	the source SHEETBODY object
tool	the tool SHEETBODY object
sheet	the resultant SHEETBODY object

API – High-Level Functions

- **intersection**

```
icode = EG_intersection(ego src, ego tool, int *nEdge,
                        ego **pFacEdg, ego *model)
icode = IG_intersection(I*8 src, I*8 tool, I*4  nEdge,
                        CPTR  pFacEdg, I*8  model)
```

Intersects the source BODY Object (that has the type SOLIDBODY, SHEETBODY or FACEBODY) with a surface or surfaces. The tool object contains the intersecting geometry in the form of a FACEBODY, SHEETBODY, SOLIDBODY or a single FACE.

src	the source BODY object
tool	the FACE/FACEBODY/SHEETBODY/SOLIDBODY tool object
nEdge	the number of EDGE objects created
pFacEdg	pointer to FACE/EDGE object pairs - 2*nEdge in len (*freeable*) can be NULL (if you don't need the data - the EDGEs are in **model**)
model	the resultant MODEL object which contains the set of WIREBODY BODY objects (this is because there may be multiple LOOPS as a result of the operation). Deleting the model invalidates the data in pFacEdg.

NOTE: The EDGE objects contained within the LOOPS have the attributes of the FACE in **src** responsible for that EDGE.

API – High-Level Functions

- **imprintBody**

```
icode = EG_imprintBody(ego src, int nObjs,
                       ego *facEdLo, ego *result)
icode = IG_imprintBody(I*8 src, I*4 nObjs,
                       I*8  facEdLo, I*8  result)
```

Imprints EDGE/LOOPs on the source BODY Object (that has the type SOLIDBODY, SHEETBODY or FACEBODY). The EDGE/LOOPs are paired with the FACEs in the source that will be scribed with the EDGE/LOOP.

src	the source BODY object
nObjs	the number of object pairs to imprint
facEdLo	list of FACE/EDGE and/or FACE/LOOP object pairs to scribe 2*nObj in len -- can be the output from **`intersect`**
result	the resultant BODY object (with the same type as the input source object, though the splitting of FACEBODY objects results in a SHEETBODY)

API – High-Level Functions

- **filletBody**

```
icode = EG_filletBody(ego src, int nEdge, ego *edges,
                      double radius, ego *result, int **maps)
icode = IG_filletBody(I*8 src, I*4 nEdge, I*8  edges,
                      R*8    radius, I*8  result, CPTR  maps)
```

Fillets the EDGEs on the source BODY Object (that has the type SOLIDBODY or SHEETBODY).

src	the source BODY object
nEdge	the number of EDGE objects to fillet
edges	list of EDGE objects to fillet – nEdge in len
radius	the radius of the fillets created
result	the resultant BODY object (with the same type as the input source object)
maps	list of Face mappings (in the result) which includes operations and an index to *src* where the Face originated – 2*nFaces in result in length (*freeable*)

API – High-Level Functions

- **chamferBody**

```
icode = EG_chamferBody(ego src, int nEdge, ego *edges,
                       ego *faces,  double dis1,
                       double dis2, ego *result, int **maps)
icode = IG_chamferBody(I*8 src, I*4 nEdge, I*8  edges,
                       I*8  faces,  R*8    dis1,
                       R*8  dis2,   I*8  result, CPTR  maps)
```

Chamfers the EDGEs on the source BODY Object (that has the type SOLIDBODY or SHEETBODY).

src	the source BODY object
nEdge	the number of EDGE objects to chamfer
edges	list of EDGE objects to chamfer - nEdge in len
faces	list of FACE objects to measure **`dis1`** from - nEdge in len
dis1	the distance from the FACE object to chamfer
dis2	the distance from the other FACE to chamfer
result	the resultant BODY object (with the same type as the input source object)
maps	list of Face mappings (in the result) which includes operations and an index to *src* where the Face originated – 2*nFaces in result in length (*freeable*)

API – High-Level Functions

- **hollowBody**

```
icode = EG_hollowBody(ego src, int nFace, ego *faces,
                      double off, int join, ego *result,
                       int **maps)
icode = IG_hollowBody(I*8 src, I*4 nFace, I*8  faces,
                      R*8    off, I*4 join, I*8  result,
                       CPTR  maps)
```

A hollowed solid is built from an initial SOLIDBODY Object and a set of FACEs that initially bound the solid. These FACEs are removed and the remaining FACEs become the walls of the hollowed solid with the specified thickness. If there are no FACEs specified then the Body is offset by the specified distance (which can be negative).

src	the source BODY object
nFace	the number of FACE objects to remove (0 performs an Offset)
faces	list of FACE objects to remove - nFace in len
off	the wall thickness (offset) of the hollowed result
join	0 - fillet-like corners, 1 - expanded corners
result	the resultant BODY object
maps	list of Face mappings (in the result) which includes operations and an index to *src* where the Face originated -- 2*nFaces in result in length (*freeable*)

Note: If **`src`** is a FACE, then **`faces`** should be a list of EDGEs and the **`result`** will be a FACE. **`maps`** in this case is not filled.

API – High-Level Functions

- **rotate**

```
icode = EG_rotate(ego src, double angle, double *axis,
                  ego *result)
icode = IG_rotate(I*8 src, R*8    angle, R*8     axis,
                  I*8  result)
```

Rotates the source Object about the axis through the angle specified. If the Object is either a LOOP or WIREBODY the result is a SHEETBODY. If the source is either a FACE or FACEBODY then the returned Object is a SOLIDBODY.

src	the source Object
angle	the angle to rotate the object through [0-360 Degrees]
axis	a point (on the axis) and a direction (6 in length)
result	the resultant BODY object (type is one greater than the input source object)

API – High-Level Functions

- **extrude**

```
icode = EG_extrude(ego src, double length, double *dir,
                   ego *result)
icode = IG_extrude(I*8 src, R*8    length, R*8     dir,
                   I*8  result)
```

Extrudes the source Object through the distance specified. If the Object is either a LOOP or WIREBODY the result is a SHEETBODY. If the source is either a FACE or FACEBODY then the returned Object is a SOLIDBODY.

src	the source Object
length	the distance to extrude
dir	the vector that is the extrude direction (3 in length)
result	the resultant BODY object (type is one greater than the input source object)

API – High-Level Functions

- **sweep**

```
icode = EG_sweep(ego src, ego spine, int mode, ego *result)
icode = IG_sweep(I*8 src, I*8 spine, I*4 mode, I*8  result)
```

Sweeps the source Object through the “spine” specified. The spine can be either an EDGE, LOOP or WIREBODY. If the source Object is either a LOOP or WIREBODY the result is a SHEETBODY. If the source is either a FACE or FACEBODY then the returned Object is a SOLIDBODY.

src	the source Object
spine	the Object used as *guide curve* segment(s) to sweep the source through
mode	sweep mode:

0 - *CorrectedFrenet*	5 - *GuideAC*
1 - *Fixed*	6 - *GuidePlan*
2 - *Frenet*	7 - *GuideACWithContact*
3 - *ConstantNormal*	8 - *GuidePlanWithContact*
4 - *Darboux*	9 - *DiscreteTrihedron*

result	the resultant BODY object (type is one greater than the input source Object)

API – High-Level Functions

- **loft**

```
icode = EG_loft(int nSection, ego *sections,
                int options,  ego *result)
icode = IG_loft(I*4 nSection, I*8  sections,
                I*4 options,  I*8  result)
```

Lofts the input Objects to create a BODY Object (that has the type SOLIDBODY or SHEETBODY).

nSection	the number of Sections in the Loft Operation
sections	list of WIREBODY or LOOP objects to Loft - nSection in len the first and last can be NODEs
options	bit flag that controls the loft: 1 - SOLIDBODY result (default is SHEETBODY) 2 - Ruled (linear) Loft (default is smooth)
result	the resultant BODY object

Note: This function may be deprecated in the future. Please use either **`EG_blend`** or **`EG_ruled`**.

API – High-Level Functions

- **blend**

```
icode = EG_blend(int nSection, ego *sections, double *rc1,
                 double *rcN,  ego *result)
icode = IG_blend(I*4 nSection, I*8  sections, R*8     rc1,
                 R*8     rcN,  I*8  result)
```

Simply lofts the input Objects to create a BODY Object (that has the type SOLIDBODY or SHEETBODY). Cubic BSplines are used. All sections must have the same number of Edges (except for NODEs) and the Edge order in each (defined in a CCW manner) is used to specify the loft connectivity.

nSection	the number of Sections in the Blend Operation interior sections can be repeated once for C1 or twice for C0
sections	list of WIREBODY or LOOP objects to Blend - nSection in len the first and last can be NODEs and/or FACEs (only one LOOP), if the first and last are NODEs and/or FACEs (and the intermediate sections are CLOSED) the result will be a SOLIDBODY otherwise a SHEETBODY will be constructed
rc1	specifies treatment* at the first section (or NULL for no treatment)
rcN	specifies treatment* at the last section (or NULL for no treatment)
result	the resultant BODY object

* for NODEs -- elliptical treatment (8 in length): radius of curvature1, unit direction, rc2, orthogonal direction; nSection must be at least 3 (or 4 for treatments at both ends)

for other sections -- setting tangency (4 in length): magnitude, unit direction

for FACEs with 2 or 3 EDGEs -- make a *Wing Tip*-like cap: zero, growthFactor (len of 2)

API – High-Level Functions

- **ruled**

```
icode = EG_ruled(int nSection, ego *sections, ego *result)
icode = IG_ruled(I*4 nSection, I*8  sections, I*8  result)
```

Produces a BODY Object (that has the type SOLIDBODY or SHEETBODY) that goes through the sections by ruled surfaces between each. All sections must have the same number of Edges (except for NODEs) and the Edge order in each is used to specify the connectivity.

nSection	the number of Sections in the Ruled Operation
Sections	A list of NODE, WIREBODY, LOOP and/or FACE objects to operate upon - nSection in len, Any FACE objects must contain only a single LOOP, Only the first and last sections can be NODEs, If the first and last sections are NODEs and/or FACEs and all WIREBODY and LOOP objects are closed, the result will be a SOLIDBODY otherwise a SHEETBODY will be constructed
result	the resultant BODY object

Note: for both **blend** and **ruled** all Loops must have their Edges ordered in a counterclockwise manner.

API – Return Codes

```
#define EGADS_CNTXTHRD -33          /* Context/thread error */
#define EGADS_READERR  -32
#define EGADS_TESSTATE -31
#define EGADS_EXISTS   -30
#define EGADS_ATTRERR  -29
#define EGADS_TOPOCNT  -28
#define EGADS_OCSEGFLT -27          /* obsolete */
#define EGADS_BADSCALE -26
#define EGADS_NOTORTHO -25
#define EGADS_DEGEN    -24
#define EGADS_CONSTERR -23          /* construction error */
#define EGADS_TOPOERR  -22
#define EGADS_GEOMERR  -21
#define EGADS_NOTBODY  -20
#define EGADS_WRITERR  -19
#define EGADS_NOTMODEL -18
#define EGADS_NOLOAD   -17
#define EGADS_RANGERR  -16
#define EGADS_NOTGEOM  -15
#define EGADS_NOTTESS  -14
#define EGADS_EMPTY    -13
#define EGADS_NOTTOPO  -12
#define EGADS_REFERCE  -11
#define EGADS_NOTXFORM -10
#define EGADS_NOTCNTX   -9
#define EGADS_MIXCNTX   -8
#define EGADS_NODATA    -7
#define EGADS_NONAME    -6
#define EGADS_INDEXERR  -5
#define EGADS_MALLOC    -4
#define EGADS_NOTOBJ    -3
#define EGADS_NULLOBJ   -2
#define EGADS_NOTFOUND  -1
#define EGADS_SUCCESS    0
#define EGADS_OUTSIDE    1          /* also -- not the same, or done */
```