

Release Specification

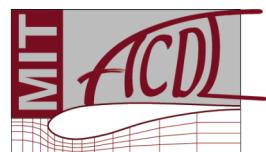


EGADS: Electronic Geometry Aircraft Design System

Bob Haimes

haimes@mit.edu

Aerospace Computational Design Lab
Department of Aeronautics & Astronautics
Massachusetts Institute of Technology



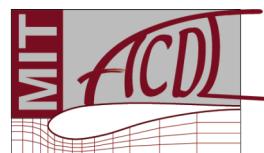
Revision 1.00 -- October 2012

AEROSPACE COMPUTATIONAL DESIGN LABORATORY

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 (32 & 64 bit):**
 - Mac OSX with **gcc**, **ifort** and/or **gfortran**
 - LINUX with **gcc**, **ifort** and/or **gfortran**
 - Windows with Microsoft Visual Studio C++ and **ifort**
 - Compiler version must match system used to build OpenCASCADE
 - No globals (but not thread-safe due to OpenCASCADE)
 - 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++
 - 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 in C/C++)



Objects

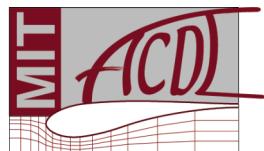


- **C Structure Definition:**

```
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**

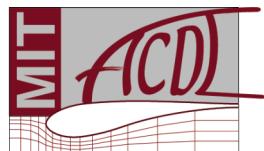
- Holds 'globals' including output level
- Start of dual-threaded list of active egos
- Pool of deleted objects



Objects -- Attribution



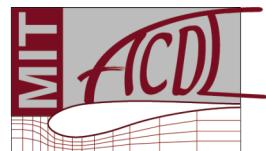
- **Attributes**
 - Are identified by a **name** (character string with no spaces or other special characters)
 - Each named attribute has a single **type**:
 - Integer
 - Real (double precision)
 - String (can have spaces and other special characters)
 - And a **length** (for Integer and Real types)
- **Objects**
 - Any Object (except for REFERENCE) may have multiple Attributes
 - Only Attributes on Topological Objects are **copied**
 - Only Attributes on Topological Objects are **persistent** -- and this is available only for ".egads" file IO.
- **SBOs and Intersection Functions**
 - Attributes on Faces will be carried through to the resultant fragments after intersections



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

Location

Curve (6)

Direction

$[x,y,z]$

$[dx,dy,dz]$

PCurve (4)

$[u,v]$

$[du,dv]$

- CIRCLE

Center

Curve (10)

Xaxis

$[x,y,z]$

Yaxis

$[dx1,dx2,dx3]$

Radius

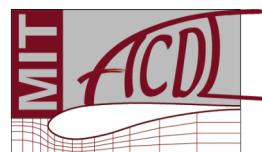
$[dy1,dy2,dy3]$

PCurve (7)

$[u,v]$

$[dx1,dx2]$

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



Objects -- PCURVE/CURVE



- **ELLIPSE**

	<u>Curve (11)</u>	<u>PCurve (8)</u>
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**

	<u>Curve (10)</u>	<u>PCurve (7)</u>
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



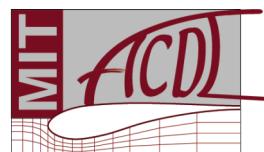
- HYPERBOLA

	<u>Curve (11)</u>	<u>PCurve (8)</u>
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)

	<u>Curve (4)</u>	<u>PCurve (1)</u>
Direction	$[dx,dy,dz]$	-
Offset		



Objects -- PCURVE/CURVE



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

BitFlag	2 - rational, 4 - periodic	
Degree		
nCP		
	<u>Curve</u>	<u>PCurve</u>
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		
	<u>Curve</u>	<u>PCurve</u>
Knots	$nKnots$	$nKnots$
ControlPts	$3*nCP$	$2*nCP$
Weights	nCP	nCP

note: **Weights** only if rational



Objects -- SURFACE



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

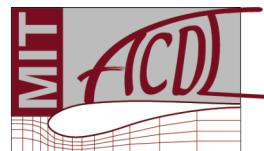
Location $[x,y,z]$
Xaxis $[dx1,dx2,dx3]$
Yaxis $[dy1,dy2,dy3]$

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

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



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 (<i>may be LeftH</i>)
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 (<i>may be LeftH</i>)
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

nCPu

vDegree

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



<u>EGADS Topological Entity</u>	<u>OpenCASCADE term</u>	<u>Geometric Entities</u>
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 **EDGES** that can be used in a non-manifold setting (for example in WireBodies)
 - Collections of **EDGES** 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 **EDGES** should be **DEGENERATE**
 - Can be **OPEN** or **CLOSED** (comes back on itself)



Objects -- Topology



- **LOOP** (with a reference **SURFACE**)
 - Collections of **EDGES** (like without a **SURFACE**) followed by a corresponding collection of **PCurves** that define the $[u,v]$ trimming on the **SURFACE**
 - **DEGENERATE EDGES** are required when the $[u,v]$ mapping collapses like at the apex of a cone (note that the **PCurve** is needed to be fully defined using the **EDGE**'s **t** range)
 - 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**

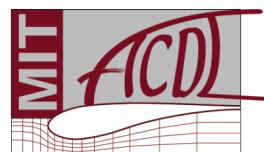
- A **SURFACE** bounded by one or more **LOOPs** with associated senses
- There may be only one outer **LOOP** (sense = 1) and any number of inner **LOOPs** (sense = -1)
- All **LOOPs** must be **CLOSED**
- If the **SURFACE** is a **PLANE**, the **LOOP(s)** must not contain any reference geometry
- If 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 ($U \times V$) 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



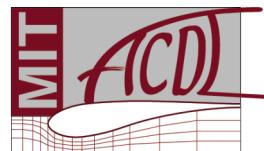
- **SHELL**
 - A collection of one or more connected **FACEs** that (if **CLOSED**) segregates regions of 3-Space
 - All **FACEs** must be properly oriented
 - **SHELLs** can be either **OPEN** or **CLOSED**
- **SOLIDBODY**
 - A collection of one or more **CLOSED SHELLs** with associated senses
 - There may be only one outer **SHELL** (sense = 1) and any number of inner **SHELLs** (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 a single **SHELL**
- **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 CURVEs at constant t increments
 - Regular Grid for SURFACEs at constant increments

- **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
- Watertight



Objects -- Tessellation



Watertight Quadrilateral FACE Treatment

- Manual
- Requires Existing Topologic Tessellation
- Must be able to Isolate 4 “sides”
 - Only single LOOPS
 - FACEs with more than 4 EDGES are analyzed to see if multiple EDGES can be treated as a single “side”
 - Currently no DEGENERATE EDGES
- 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 -- Others

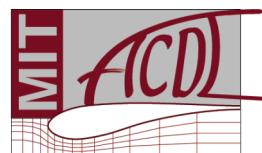


- **TRANSFORM**

- Used when copying Objects to change the root position and orientation

- **REFERENCE**

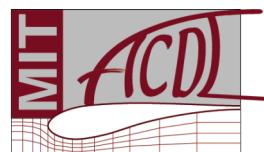
- Allows of the management of Objects that refer to other Objects (so that deletion does not invalidate the data)
 - This is an internal Object and is not usually seen by the EGADS programmer.



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.

- **free**

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

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

- **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**

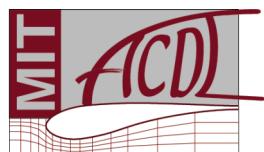
```
icode = EG_close(ego context)
icode = IG_close(I*8 context)
```

Cleans up and closes the **CONTEXT**

- **revision**

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

Returns the version information for EGADS





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:

1 - Don't split closed and periodic entities

name: Load by extension

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



- **getTransform**

```
icode = EG_getTransform(ego oform, double *xform)
icode = IG_getTransform(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 factor).



API -- Utility & IO Functions



- **copyObject**

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

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

object the input object (3D geometry or topology)
oform the transformation object (may be NULL for a strict copy)
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, EDGE, 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, EGDE, LOOP, FACE, SHELL, BODY, MODEL

mtype PCURVE/CURVE
LINE, CIRCLE, ELLIPSE, PARABOLA, HYPERBOLA, TRIMMED,
BEZIER, BSPLINE, OFFSET
SURFACE
PLANE, SPHERICAL, CYLINDRICAL, REVOLUTION, TORIODAL,
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



Additional Memory Functions for the FORTRAN Bindings

- **alloc**

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

Allocates a block of memory

- **calloc**

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

Allocates a zero fills a block of memory

- **realloc**

```
icode = IG_realloc(CPTR ptr, I*4 nbytes)
```

Reallocates a block of memory





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

len the number of integers or reals (ignored for strings)

ints the integers for ATTRINT

reals the floating point data for ATTRREAL

string the character string for an ATTRSTRING type



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 or ATTRSTRING

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

string the returned character string for an ATTRSTRING type

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



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 or ATTRSTRING

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

string the returned character string for an ATTRSTRING type

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



API -- Attribution

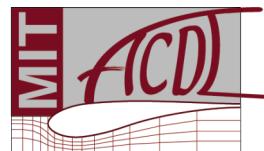


- **attributeDup**

```
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



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

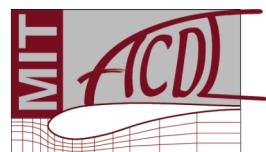
SURFACE

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*).



API -- Geometry

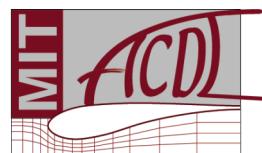


- **makeGeometry**

```
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)



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 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

eval the returned position, 1st and 2nd derivatives (length):

	<u>PCurve (6)</u>	<u>Edge -or- Curve (9)</u>	<u>Face -or- Surface (18)</u>
Position	$[u, v]$	$[x, y, z]$	$[x, y, z]$
1st Derivative	$[du, dv]$	$[dx, dy, dz]$	$[du_x, du_y, du_z]$ $[dv_x, dv_y, dv_z]$
2nd Derivative	$[du^2, dv^2]$	$[dx^2, dy^2, dz^2]$	$[du_x^2, du_y^2, du_z^2]$ $[d(uv)_x, d(uv)_y, d(uv)_z]$ $[dv_x^2, dv_y^2, dv_z^2]$



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 EGDE/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)



API -- Geometry

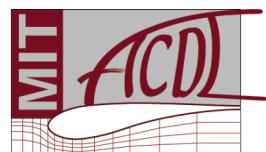


• approximate

```
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 object created by approximating a BSpline. If the tolerance is zero for a SURFACE then the data is fit.

context	the CONTEXT object used to place the result
mDeg	is the maximum degree used for the approximation [3-8] Note: fits are always cubic.
tol	is the tolerance to use for the BSpline approximation procedure
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 of coordinates.
xyz	the data to fit (3 times the number of points in length)
geo	the returned approximated (or fit) BSpline resultant object



API -- Geometry



- **otherCurve**

```
icode = EG_otherCurve(ego surface, ego iCrv, double tol,  
                      ego *oCrv)  
icode = IG_otherCurve(I*8 surface, 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 curve is a 3D CURVE (and *vice versa*).

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





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 **CURVE**, **EDGE**, **SURFACE** or **FACE**
bspline the returned approximated resultant **BSPLINE** object





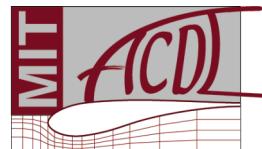
API -- Topology

- **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

- **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.

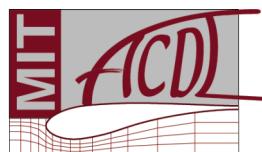
object topological object (all except MODEL)
tol the tolerance used to define closure

- **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. This also creates any required NODEs, EDGES and LOOPS.

object either a LOOP (for a planar *cap*) or a SURFACE with $[u,v]$ bounds
mtype is either SFORWARD or SREVERSE
for LOOPS you may want to look at the orientation using **getArea**
data may be NULL for LOOPS but must be the limits for a SURFACE
face the resultant returned topological FACE object



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

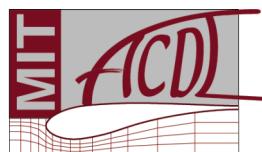
sphere (4): [x,y,z] of center then radius

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 is the reference topological object or NULL
 - this 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, EGDE, 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*)





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

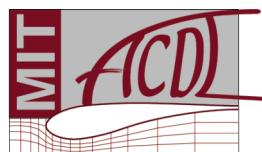
- **getBoundingBox**

```
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, FACE, SHELL, BODY or MODEL
data the data returned (must be declared to at least 14 doubles):
 volume, surface area (length for EDGE)
 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. Note: topological objects in different bodies will have different object pointers.



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

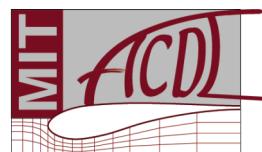


- **getEdgeUV**

```
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 if the EDGE is found the the FACE twice. This uniquely specifies which position to use when the FACE closes on itself.
t the parametric value to use for the evaluation
uv the resulting $[u,v]$ evaluated at t .



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

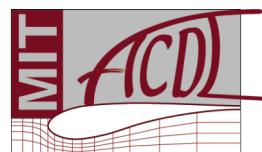


- **makeTessBody**

```
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.



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



- **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 `getBodyTopos`.
- 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.



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 [getBodyTopos](#).
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 (1 bias)
ptric returned pointer to neighbor information



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

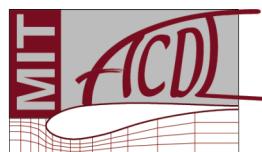


- **getPatch**

```
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 $n1 \times n2$ 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.

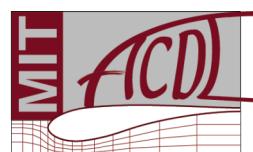
tess	the TESSELLATION object (not on WIREBODIES)
eIndex	the EDGE index (1 bias).
vIndex	the Vertex index in the EDGE (2 - nVert-1)
t	the new parameter value on the EDGE for the point

• 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-based 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.



API -- High-Level Functions



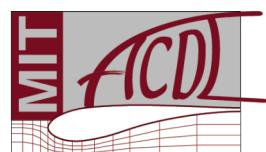
- **solidBoolean**

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



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. The tool object contains the surface in the form of a FACEBODY (BODY object) or a single FACE.

src the source BODY object
tool the FACE/FACEBODY 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 this 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).

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 nEdge,  
                      ego *facEdg, ego *result)  
icode = IG_imprintBody(I*8 src, I*4 nEdge,  
                      I*8 facEdg, I*8 result)
```

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

src the source BODY object
nEdge the number of EDGE objects to imprint
facEdg list of FACE/EDGE object pairs to scribe - 2*nEdge in len
 can be the output from **intersect**
result the resultant BODY object (with the same type as the input source object)



API -- High-Level Functions

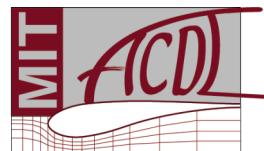


- **filletBody**

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

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)



API -- High-Level Functions



- **chamferBody**

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

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)



API -- High-Level Functions



• hollowBody

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

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.

src the source BODY object
nFace the number of FACE objects to remove
faces list of FACE objects to fillet - nFace in len
off the wall thickness (offset) of the hollowed result
join 0 - fillet-like corners, 1 - expanded corners
result the resultant SOLIDBODY object



API -- High-Level Functions

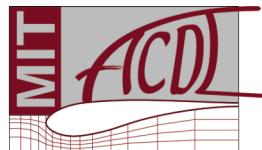


- **rotate**

```
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 edge, ego *result)  
icode = IG_sweep(I*8 src, I*8 edge, I*8 result)
```

Sweeps the source Object through the EDGE Object specified. 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
edge the EDGE Object used as a *guide curve* (with limits) to sweep through
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



API -- Return Codes



```
#define EGADS_OCSEGFLT -27      /* OpenCASCADE Seg Fault */
#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 */
```

